Ministério da Educação
Universidade Federal de Ouro Preto
Escola de Minas
Departamento de Engenharia de Produção

Um algoritmo simheurístico para a resolução do Problema de Flow Shop Permutacional multiobjetivo

Lucas César Barbosa

Ouro Preto MG
2022

Lucas César Barbosa

Um algoritmo simheurístico para resolução do Problema de Flow Shop Permutacional multiobjetivo

Relatório Final, referente ao período de setembro de 2021 à setembro de 2022, apresentado à Universidade Federal de Ouro Preto, como parte das exigências do programa de iniciação cientifica / PIP.

Orientador: Prof. Dr. Aloisio de Castro Gomes Junior

Coorientador: Prof. Dr. Helton Cristiano Gomes

Coorientadora: Naiara Helena Vieira

Ouro Preto – MG

Setembro 2022

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA DE PRODUCAO, ADMINISTRACAO E ECON

FOLHA DE APROVAÇÃO

Lucas César Barbosa

Um Algoritmo Simheurístico para a Resolução do Problema de Flowshop Permutacional Multiobjetivo

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de bacharel em Engenharia de Produção.

Aprovada em 15 de setembro de 2022

Membros da banca

[Doutor] - Aloísio de Castro Gomes Júnior - Orientador(a) - (Universidade Federal de Ouro Preto)
[Doutor] - Helton Cristiano Gomes - Coorientador - (Universidade Federal de Ouro Preto)
[Doutor] - Magno Silvério Campos - (Universidade Federal de Ouro Preto)
[Doutor] - Irce Fernandes Gomes Guimarães - (Universidade Federal de Ouro Preto)
[Mestranda] - Naiara Helena Vieira - (Universidade Federal de Ouro Preto)

Aloísio de Castro Gomes Júnior, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 24/10/2022

Documento assinado eletronicamente por **Aloisio de Castro Gomes Junior**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 24/10/2022, às 14:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0416624** e o código CRC **4FC34BE4**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591540 - www.ufop.br

Dedico este trabalho aos meus pais, minha irmã e minha namorada que sempre me incentivaram e apoiaram.

Agradecimentos

Agradeço a minha família e minha namorada por todo o apoio e suporte durante toda a graduação.

Ao meu orientador Professor Doutor Aloísio de Castro Gomes Júnior, a minha coorientadora Naiara Helena Vieira e coorientador Professor Doutor Helton Cristiano Gomes pelos ensinamentos durante o desenvolvimento deste trabalho.

Aos meus amigos de graduação que sempre estiveram comigo.

Resumo

Este trabalho apresenta uma simheurística para resolução do problema de *Flow Shop* permutacional multiobjetivo (PFSP-MO). Este que se caracteriza como um problema comum dentro da indústria atual onde *jobs* devem ser executados em todas as máquinas na mesma ordem visando a otimização de dois ou mais objetivos, como o *makespan*, atraso total, antecipação total, atraso por máquina e etc. Um ponto que pode-se destacar neste tipo de problema é a existência de fatores de atraso, como quebras de máquinas, paradas para manutenção, tempos de *setup*, entre outros. Para simular estes atrasos foi inserido um parâmetro estocástico no método. Para a resolução do problema apresentado, desenvolveu-se um algoritmo genético (*Nondominated sorting algorithm II* (NSGA-II)) na linguagem *Python*, com objetivo de minimizar o *makespan*, atraso total e antecipação total. A simheurística desenvolvida foi aplicada em 120 instâncias disponíveis na literatura. O algoritmo desenvolvido apresentou bons resultados para o problema, conseguindo entregar soluções diversas e com bons valores para cada um dos objetivos.

Palavras-chave: *Flow Shop* Permutacional, Simheurística, Multiobjetivo, *Non-dominated Sorting Algorithm II*

Abstract

This work presents a simheuristic to solve the multi-objective permutational Flow Shop problem (PFSP-MO). This is characterized as a common problem within the current industry where jobs must be executed on all machines in the same order in order to optimize two or more objectives, such as makespan, total delay, total anticipation, delay per machine, etc. A point that can be highlighted in this type of problem is the existence of delay factors, such as machine breakdowns, maintenance stops, setup times, among others. To simulate these delays, a stochastic parameter was inserted in the method. To solve the problem presented, a genetic algorithm (Nondominated sorting algorithm II (NSGA-II)) was developed in Python language, with the objective of minimizing the makespan, total delay and total anticipation. The developed symheuristic was applied to 120 instances available in the literature. The developed algorithm presented good results for the problem, managing to deliver diverse solutions with good values for each of the objectives.

Keywords: Flow Shop Permutacional, Simheuristic, Multi-objective, Non-dominated Sorting Algorithm II

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

PFS	Problema Flow Shop
PFSP	Problema Flow Shop Permutacional
PFSP-MO	Problema Flow Shop Permutacional Multiobjetivo
AG	Algoritmo Genético
NSGA	Nondominated Sorting Genetic Algorithm
NSGA-II	Nondominated Sorting Genetic Algorithm II
AEs	Algoritmos Evolucionários
FIFO	First-In-First-Out
SA	Simulated Annealing
ILS	Iterated Local Search
VNS	Variable Neighborhood Search
NEH	Nawaz-Enscore-Hoam
CSW	Crow Search Algorithm
PCV	Problema do Caixeiro Viajante
ORND	Ordenação Rápida Não Dominada
DM	Distância de Multidão
FP	Fronteira Pareto
PMX	Partially Mapped Crossover
MO	Multiobjetivo

Sumário

1 Introdução

Para atingir seus objetivos e aplicar adequadamente seus recursos, as empresas não produzem ao acaso. Elas precisam planejar antecipadamente e controlar de forma adequada sua produção (CHIAVENATO, 2008, p. 23). O ambiente industrial passou durante as últimas décadas por grandes avanços, os quais tornaram cada vez mais complexas as tomadas de decisão dentro dos ambientes produtivos.

A programação da produção (*scheduling*) representa a tomada de decisão que visa alocação de recursos às tarefas em um dado período de tempo, tendo como finalidade otimizar um ou mais objetivos (PINEDO, 2012). Uma programação da produção que atenda às necessidades do ambiente produtivo é fundamental para a indústria atual. Arenales et al. (2015) afirmam que as principais decisões envolvidas no nível operacional são a designação de tarefas (*jobs*) à máquinas e a programação das tarefas em cada máquina. Um job representa uma sequência de um determinado número de tarefas ou operações necessárias para a fabricação de um produto (PINEDO, 2012).

Normalmente, em um ambiente produtivo existem uma quantidade maior de ordens de produção que estações de trabalho, ocasionado filas e uma alta demanda pelos recursos produtivos transformadores (CARDOSO, 2021). O sequenciamento das ordens de produção é uma tarefa complexa.

Para auxiliar na tomada de decisão e definir a melhor ordem para a execução dos *jobs* é empregada, dentre outras ferramentas, os métodos matemáticos da pesquisa operacional, que buscam soluções ótimas para os problemas.

Dentro da indústria de manufatura e serviços existem alguns ambientes de produção, sendo um deles o *Flow Shop*. Segundo LAGE (2019) o Problema de *Flow Shop* (PFS) consiste em *m* máquinas diferentes onde cada *job* precisa ser processado por cada uma das máquinas uma única vez, todas as tarefas tem a mesma rota de processamento e uma tarefa não pode iniciar sua manufatura na máquina *m* até que seu processamento na máquina *m-1* tenha sido concluídos.

Uma variante do PFS é o problema de *Flow Shop* permutacional (PFSP), onde todos os *jobs* devem ser executados em todas as máquinas. O PFSP é um problema bem conhecido de otimização combinatória e até onde se sabe a maioria dos PFSP envolvem múltiplos objetivos conflitantes como *makespan*, atraso total, atraso por máquina e assim por diante (LI; MA, 2016). Nos últimos anos, o PFSP passou a receber cada vez mais atenção devido ao seu papel fundamental dentro da produção, *designs* computacionais, distribuição, transportes, processamento de informações e comunicações (ABDEL-BASSET; MANOGARAN; EL-SHAHAT, 2018).

Os problemas do mundo real, em sua grande maioria, são compostos por problemas

multiobjetivos, normalmente esses objetivos são conflitantes, a melhora de um determinado objetivo piora o resultado de outro. Durante as últimas décadas, várias técnicas de otimização foram desenvolvidas para o problema de *Flow Shop* permutacional multiobjetivo (PFSP-MO), dentre elas pode-se destacar os Algorítimos Genéticos (AG), que dada a sua aplicação bem sucedida para problemas multiobjetivo, diversos pesquisadores desenvolveram AG para resolução do PFSP-MO (LI; MA, 2016).

O PFS com três ou mais máquinas foi classificado por Garey e Johnson (1977) como *NP-hard*, desta forma, encontrar boas soluções para o PFS é uma tarefa difícil (RIFAI et al., 2020).

Métodos matemáticos tem grande dificuldade na resolução de problemas NP-difíceis, com tempo de processamento grande. Problemas NP-difíceis tornam o uso dos métodos matemáticos inviáveis. Segundo Goldbarg, Goldbarg e Luna (2016), diante da escassez de respostas dos métodos exatos, durante os últimos anos métodos aproximativos e mais eficientes foram o foco do desenvolvimento na área para solução de problemas *NP-hard* de maior porte. Os métodos retratados por Goldbarg, Goldbarg e Luna (2016) são os heurísticos.

Uma heurística visa encontrar boas soluções para problemas, não necessariamente sendo uma solução ótima global, mas entregando como resultado um ótimo local que é um resultado melhor que uma escolha arbitrária ou que não se baseia em nenhum outro método, sendo executada em um tempo computacional aceitável (GOLDBARG; GOLDBARG; LUNA, 2016). Segundo Hillier e Lieberman (2013) as heurísticas fazem uso de ideias simples para procurar boas soluções para um problema.

Um problema citado por Hillier e Lieberman (2013) é de que cada heurística atende especificamente a um tipo de problema, sendo necessária a adaptação da mesma para cada problema. A adaptação das heurísticas se torna então um ponto problemático, que demanda tempo para ser solucionado quando se mudam os problemas trabalhados. Porém, nos últimos anos, surgiram as meta-heurísticas. Segundo Goldbarg, Goldbarg e Luna (2016, p. 72) as "meta-heurísticas se tratam de uma arquitetura geral de regras que, formada a partir de um tema em comum, pode servir de base para o projeto de uma ampla gama de heurísticas computacionais". As meta-heurísticas permitem uma fácil adaptação para diversos problemas.

Uma meta-heurística de busca local se caracteriza por efetuar buscas ao redor de uma solução inicial, porém apresentando formas que podem as remover de pontos de ótimos locais, assim podendo percorrer outros ótimos locais e até mesmo o chegar ao ótimo global. Elas fazem uso de heurísticas que efetuam as buscas locais dentro de uma solução previamente gerada.

Os Algoritmos genéticos (AGs), introduzidos por Holland (1975), têm sido uma das meta-heurísticas de maior sucesso para a solução de problemas de otimização combinatória (GOLDBARG; GOLDBARG; LUNA, 2016). Os algoritmos genéticos se enquadram dentro do grupo chamado de Algoritmos Evolucionários (AEs). Os AEs, segundo Goldbarg, Goldbarg

e Luna (2016), buscam a representação do problema com uma representação em forma de informação genética, esta é processada e submetida ao processo de evolução artificial. Por fim um processo final devolve a informação em forma de uma solução para o problema.

Para a solução de problemas multiobjetivos o *nondominated sorting genetic algorithm II* (NSGA-II) se destaca. Este algoritmo foi desenvolvido por Deb et al. (2002), trata-se de um algoritmo elitista de busca populacional que classifica as soluções por não dominância utilizando o conceito ótimo de Pareto.

Assim visto a grande aplicabilidade e utilidade dos métodos meta-heurísticos o presente trabalha visa aplicar o NSGA-II para solução de um problema de *Flow Shop* permutacional multiobjetivo, buscando minimizar três objetivos conflitantes o *makespan*, atraso total e antecipação total.

Em problemas do mundo real, eventos inesperados podem ocorrer, como quebras de máquinas, atraso com insumos e mudanças de *setup* das máquinas. Assim para simular um ambiente com eventos incertos um parâmetro estocástico é inserido. O presente trabalho então busca a resolução de um PFSP-MO estocástico.

O objetivo principal do trabalho é aplicar uma simheurística com o algoritmo genético NSGA-II para resolução do PFSP-MO estocástico, fazendo uso de instâncias adaptadas de (TAILLARD, 1993) por (VIEIRA, 2022).

Os objetivos específicos do trabalho são estudar e aplicar simheurísticas, estudar algoritmos genéticos e desenvolver um algoritmo genético (NSGA-II) para resolução do PFSP-MO estocástico.

2 Referencial teórico

Este capítulo apresenta um revisão da literatura a respeito de problemas de programação da produção com ênfase no PFS, meta-heurísticas multiobjetivos, algoritmos genéticos e sim-heurísticas e detalha-se o NSGA-II, utilizado no presente trabalho para gerar as soluções do problema.

2.1 Meta-heurísticas

Encontrar soluções exatas para problemas *NP-Hard* é um grande desafio, exigindo grande poder de processamento e memória dos computadores (GOLDBARG; GOLDBARG; LUNA, 2016). As heurísticas, que começaram a ser desenvolvidas no final da década de 1950 buscam solucionar esses problemas, no entanto algoritmos exatos possuem alta demanda de tempo computacional (ARENALES et al., 2015). Os métodos heurísticos encontram boas soluções para os problemas, porém não garantem um ótimo global como solução, mas entregam os seus resultados em tempo computacional hábil (HILLIER; LIEBERMAN, 2013; ARENALES et al., 2015).

As meta-heurísticas são aplicadas para problemas de minimização ou maximização, e sujeitas a um conjunto de restrições.

Goldbarg, Goldbarg e Luna (2016) classificam as meta-heurísticas em função da estratégia de obtenção das soluções:

(a) Construtivas: Nesta estratégia as soluções são classificadas em ordem crescente podendo ou não serem melhoradas durante a execução do algoritmo, alguns exemplos de métodos são o GRASP e Colônia de Formigas.

(b) Evolutivas: Essa por sua vez consiste na criação de uma solução a partir de outras soluções já criadas, os algoritmos evolucionários são os exemplos mais claros dessa categoria, como os AG, Algoritmos Meméticos, Simbióticos e outros.

(c) De Decomposição: As soluções nestas meta-heurísticas são organizadas de forma que o problema principal é subdividido em problemas de solução mais fácil, alguns exemplos de meta-heurísticas dessa classe são a heurística de Dantzig-Wolfe e decomposição de Benders.

(d) De Informação Compartilhada: Nesta estratégia as soluções ficam organizadas a partir de um grupo de agentes ou métodos que constroem ou modificam as soluções ao longo do processo, a depender do tipo de construção ou compartilhamento de informações uma

classe de meta-heurísticas pode fazer uso dessa estratégia, alguns exemplos de algoritmos são Colônia de Formigas, Nuvem de Partículas e Algoritmos Transgenéticos.

Conforme citado anteriormente as meta-heurísticas são a forma mais utilizada para se resolver o PFSP. A aplicação de meta-heurísticas se deu, inicialmente, para problemas mono-objetivos e com seu grande sucesso seu uso foi ampliado para problemas multiobjetivos.

2.2 Fronteira pareto

A fronteira pareto (FP) é responsável por armazenar as soluções que em pelo menos um dos objetivos é melhor que qualquer outra dentro do grupo de soluções. Esse grupo também pode ser chamado de soluções pareto ótimas. O conceito de dominância de pareto é responsável por criar conjuntos de solução ótimas de pareto, que compõem a referida fronteira, onde nenhum dos indivíduos é dominado por qualquer outro dentro do conjunto de soluções (COTA et al., 2022).

Para Li e Ma (2016) existem quatro conceitos importantes para problemas multiobjetivos:

1. Dominância: uma solução $x \in \omega$ domina uma outra solução $x^* \in \omega$ se e somente se:

$$\forall i \in \{1, ..., k\} : f_i(x) \leq f_i(x^*) \tag{2.1}$$

$$\wedge \exists j \in \{1, ..., k\} : f_j(x) \leq f_j(x^*) \tag{2.2}$$

2. Otimalidade de pareto: uma solução $x^* \in \omega$ é considerada pareto ótima se não existir um $x \in \omega$ tal que $f(x)$ domine $f(x^*)$.

3. Conjunto ótimo de pareto: as soluções incluídas dentro do conjunto ótimo de pareto são os chamados indivíduos não dominados.

4. Fronteira pareto: Em problemas multi-objetivos um conjunto de soluções chamadas ótimas compões a chamada frente de pareto. O presente trabalho tem três objetivos a serem avaliados dentro da função objetivo do problema a minimização do makespan equação 2.3, do atraso total equação 2.4 e do antecipação total equação 2.5.

$$C_{max} = maxC_i \tag{2.3}$$

$$T_{total} = \sum_{i \in N} T_i \tag{2.4}$$

$$E_{total} = \sum_{i \in N} E_i \tag{2.5}$$

2.3 Problemas de programação da produção

Para Arenales et al. (2015) as principais decisões envolvidas no nível operacional são: designação de *jobs* a máquinas e programação das tarefas em cada máquina definindo a sequência de processamento das tarefas e seu momento de início e término de processamento.

Os problemas relativos a programação da produção aparecem quando produtos devem ser fabricados e existe uma capacidade limitada de recursos produtivos que podem ser destinados a essas atividades, sendo esses recursos, por exemplo, máquinas, equipamentos e mão de obra especializada (RIFAI et al., 2020). Os problemas de programação da produção tratam da alocação de recursos para tarefas buscando a minimização ou maximação de um ou mais objetivos (NOGUEIRA et al., 2014).

A importância desses problemas faz com que sejam amplamente estudados na literatura. Haja visto que estão presentes em diversas áreas como: indústrias química, metalúrgica e têxtil (NOGUEIRA et al., 2014).

Arenales et al. (2015) citam que as principais medidas de desempenho, relativas a avaliação da qualidade de uma programação de produção, são:

- *Makespan*: representa o tempo total de processamento das tarefas, ou seja, o instante de término do da última tarefa na última máquina.
- tempo total de fluxo: é a soma dos instantes de término do processamento das tarefas.
- Atraso máximo: o maior atraso dentro os processamentos.
- Atraso total: a soma de todos os atrasos, ou seja, a soma das diferenças entre a data de término real e a data de entrega projetada.
- Antecipação: representa a antecipação nas entregas dos produtos, ou seja, a diferença entre o tempo de término da execução dos jobs e sua data esperada de entrega. Quanto menor esse tempo, menor será o número de produtos em estoque esperando a entrega para o cliente final.
- número de tarefas atrasadas: soma das tarefas atrasadas.

Normalmente, dentro da indústria existe a necessidade de se otimizar mais de um objetivo, sendo estes muitas vezes conflitantes. Grimme e Lepping (2011) ressaltam que um ambiente produtivo é avaliado de acordo com diversos critérios mutuamente, como o *makespan*, atraso total, antecipação total ou número total de *jobs* atrasados.

Arenales et al. (2015) e LAGE (2019) apresentam em suas obras os principais ambientes de máquinas:

(a) Uma máquina: o ambiente com uma máquina é considerado o mais simples. Uma única máquina é responsável por trabalhar todas as tarefas e entregar os produtos.

(b) Máquinas paralelas: este ambiente pode ser dividido em três subcategorias segundo Arenales et al. (2015), máquinas idênticas, máquinas uniformes e não relacionais. O caso de máquinas idênticas é o mais simples deles, um determinado job pode ser executado em qualquer uma das *m* máquinas. O caso de máquinas uniformes apresenta uma uniformidade entre os tempos de preparação e produção das máquinas. Por fim, o de máquinas não relacionais onde os tempos de preparação e produção são diferentes.

(c) *Job Shop*: é um ambiente onde existem *n jobs* e *m* máquinas, a ordem de processamento dos *jobs* nas máquinas segue um ordem específica, porém não necessariamente um *job* deve ser executado em todas as máquinas.

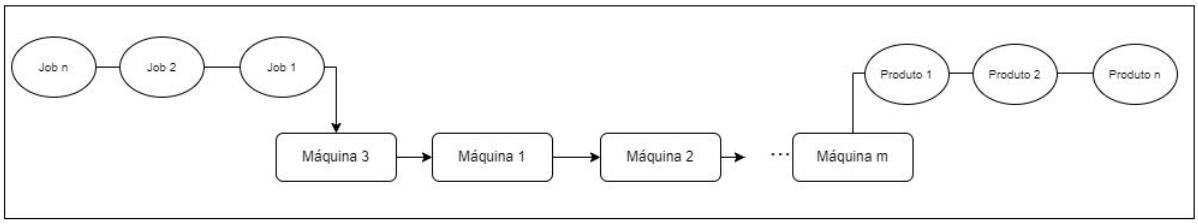
(d) *Flow Shop*: O *Flow Shop* é um ambiente que possui *m* máquinas e um *job* deve ser executado em todas as máquinas na mesma ordem, da máquina 1 até a *m*. Um caso particular ocorre quando os *jobs* tem a mesma sequência em todas as máquinas, assim tem-se um Problema de *Flow Shop* Permutacional.

A próxima seção apresenta e detalha o PFS, que é o foco do presente trabalho.

2.4 Flow Shop Permutacional

O PFS é um dos problemas de programação da produção mais estudados na literatura (RUIZ; STüTZLE, 2006). No PFS os *n jobs* são processados por todas as *m* máquinas, seguindo a mesma ordem de processamento. Quando em um PFS for adotado o sistema *First-In-First-Out* (FIFO), primeiro a entrar é o primeiro a sair, este é chamado de Problema de Flow Shop Permutacional (PINEDO, 2012). A figura 1 ilustra o PFSP.

Figura 1 – Representação problema flow shop permutacional.



Fonte: Autor.

Existem três formas para se resolver um PFSP: algoritmos tradicionais, algoritmos heurísticos e algoritmos meta-heurísticos, sendo os meta-heurísticos os mais utilizados durante os últimos anos (ABDEL-BASSET; MANOGARAN; EL-SHAHAT, 2018).

Na literatura o PFSP é trabalhado, na maior parte dos estudos, como um problema mono-objetivo. Segundo Abdel-Basset, Manogaran e El-Shahat (2018) o objetivo mais comum dentro do PFSP é a minimização do *makespan*. Diversas heurísticas e meta-heurísticas são utilizadas para resolver o PFSP e PFSP-MO.

Rifai et al. (2020), em seu trabalho, desenvolvolveram um algoritmo híbrido que utiliza um AG e o *Simulated Annealing* (SA) para resolução de um problema de *Flow Shop* Permutacional Reentrante, onde máquinas podem processar um *job* mais de uma vez. Os objetivos minimizados foram o *makespan*, o tempo médio de conclusão e o atraso total.

Ferone et al. (2019) desenvolveram um algoritmo *Iterated Local Search* (ILS) que inicia uma solução de forma tendenciosa, buscando soluções melhores. O problema tratado é dividido em duas partes, sendo a primeira composta por diversas fábricas com configurações *Flow Shop* e a segunda consiste na união de cada um deles para a formação de um produto final, tendo como objetivo a minimização do *makespan* do processo.

Eddaly, Jarboui e Siarry (2016) desenvolveram um algoritmo de partículas híbridas para resolver o PFSP, para melhorar as soluções é utilizado um algoritmo de busca local. O objetivo foi minimizar o *makespan*.

Rahman, Sarker e Essam (2015) desenvolveram um AG para resolução do PFSP, que teve como objetivo a minimização da soma dos custos de instalação e manutenção.

Xie et al. (2014) desenvolveram um algoritmo de ensino e aprendizagem que combina um *Variable Neighborhood Search* (VNS) para melhoria das soluções geradas, de forma rápida, por meio de uma busca local. Ainda é implementada uma nova busca nas soluções utilizando um SA para melhora das soluções. Os objetivos minimizados foram o *makespan* e o atraso máximo.

Li e Ma (2016) propuseram um algoritmo memético para a resolução do PFSP-MO, como heurística construtiva para as soluções foi utilizada a heurística de *Nawaz-Enscore-Hoam* (NEH). Esta heurística seleciona soluções de forma aleatória dentro da população inicial e efetua uma busca local com objetivo de melhorar a solução inicial aleatória, a partir disso são realizada buscas globais na população de soluções. Os objetivos minimizados foram o *makespan* e atraso total.

Abdel-Basset, Manogaran e El-Shahat (2018) implementaram um novo algoritmo que une o *Whale Optimization Algorithm* a uma estratégia de busca local para resolução PFSP, o NEH é aplicado buscando uma melhoria de desempenho do algoritmo. O objetivo minimizado foi o *makespan*.

Huang et al. (2019) desenvolveram um *Crow Search Algorithm* (CSW), como método para criação da população inicial foi utilizado o NEH, também foi utilizado um método para executar uma busca local para melhora das soluções iniciais. O VNS e o SA foram utilizados como métodos de buscas locais.O objetivo foi a minimização do *makespan*.

2.4.1 Modelo Matemático do PFSP

Arenales et al. (2015) apresentam um modelo matemático para o PFSP, com o objetivo de minimizar o *makespan* (C_{max}). Assumindo *j* o índice que indica a posição na sequência de tarefas.

A seguir são descritos os parâmetros do modelo.

p_{ik} = tempo de processamento da tarefa *i* na máquina *k*.

As variáveis são definidas:

s_{kj} = instante de início de processamento da tarefa na posição *j* na máquina *k*.

$$Z_{ij} = \begin{cases} 1 \text{ se a tarefa } i \text{ é designada à posição } j \\ 0 \text{ caso contrário} \end{cases}$$

A seguir está apresentado o modelo:

$$minC_{max} = s_{mn} + \sum_{i=1}^{n} p_{im} Z_{in} \tag{2.6}$$

$$\sum_{j=1}^{n} Z_{ij} = 1, i = 1, ..., n \tag{2.7}$$

$$\sum_{i=1}^{n} Z_{ij} = 1, j = 1, ..., n \tag{2.8}$$

$$s_{1j} + \sum_{i=1}^{n} p_{i1} Z_{ij} = s_{1,j+1}, j = 1, ..., n-1 \tag{2.9}$$

$$s_{11} = 0 \tag{2.10}$$

$$s_{k1} + \sum_{i=1}^{n} p_{ik} Z_{i1} = s_{k+1,1}, k = 1, ..., m-1 \tag{2.11}$$

$$s_{kj} + \sum_{i=1}^{n} p_{ik} Z_{ij} \leq s_{k+1,j}, j = 2, ..., m-1 \tag{2.12}$$

$$s_{kj} + \sum_{i=1}^{n} p_{ik} Z_{ij} \leq s_{k,j+1}, j = 1, ..., m-1, k = 2, ..., m \tag{2.13}$$

$$s \in R_{+}^{nm}, Z \in B^{nm} \tag{2.14}$$

A equação 2.6 representa a minimização do *makespan*. As restrições 2.7 garantem que cada *job i* esta alocado a uma única posição, as restrições 2.8 garantem que cada posição *i* está

ligada a um único *job*. As equações 2.9 obrigam que um *job* na posição j inicie seu processamento na máquina 1 depois que o seu *job* predecessor tenha passado por esta máquina. As equações 2.10 estabelecem que a primeira tarefa da sequência inicie seu curso na máquina 1 no instante igual a 0. As restrições 2.11 definem que obrigatoriamente o primeiro *job* na sequência seja executado imediatamente na próxima máquina k+1, se a sua execução tenha sido finalizada na máquina corrente k. As equações 2.12 estabelecem que um *job* na posição j não pode ser iniciado na próxima máquina k+1 antes de sua finalização na máquina corrente k. As restrições 2.13 estabelecem que um *job* na posição j+1 não pode iniciar em uma máquina k antes da finalização do *job* na posição j na mesma máquina k tenha sido completado. Por fim, a 2.14 determinam o domínio das variáveis.

2.4.2 PFSP estocástico

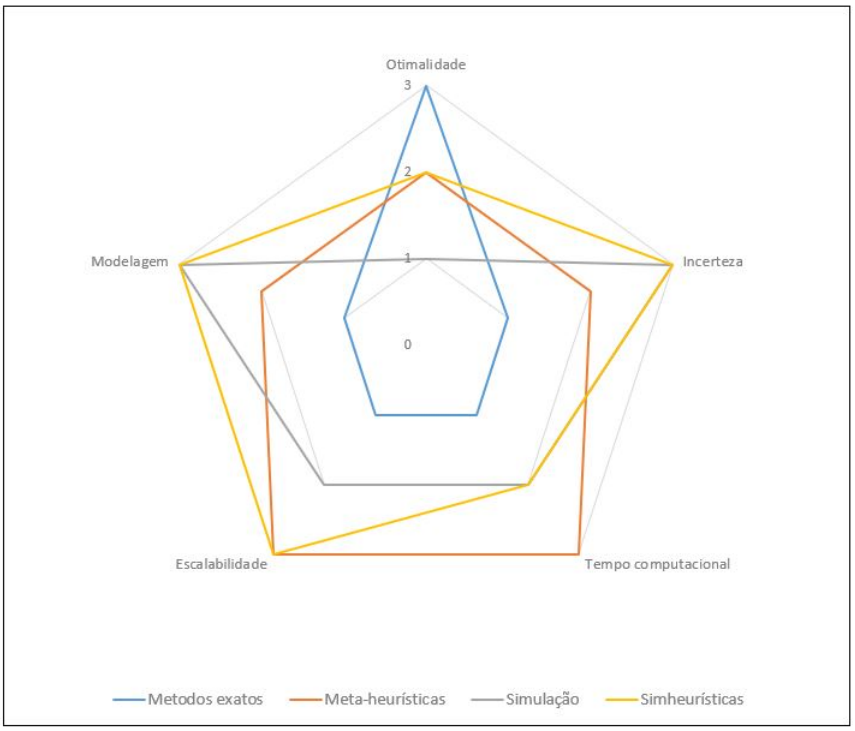
O PFSP-MO se caracteriza como um problema que pode assumir uma natureza estocástica ou determinística (COTA et al., 2022). Normalmente os problemas são caracterizados como determinísticos, porém a incerteza está sempre presente tornando problemas determinísticos versões simplificadas de problemas reais (JUAN et al., 2015).

Em problemas do mundo real eventos inesperados podem ocorrer, como quebras de máquinas, atraso com insumos e mudanças no tempo de preparação das máquinas. Assim para simular um ambiente com eventos incertos um parâmetro estocástico pode ser inserido para simular um problema mais proximo da realidade. O uso do parâmetro estocástico junto a uma meta-heurística busca manter os pontos positivos de ambas as abordagens para resolução de problemas, criando assim as simheurísticas. As simheurísticas permitem que os modeladores lidem com a incerteza da vida real de maneira natural, integrando a simulação (em qualquer uma de suas variantes) em uma estrutura orientada por meta-heurísticas (JUAN et al., 2015).

Simulações permitem a construção e reprodução de sistemas complexos de forma natural, normalmente não exigem nenhuma implementação matemática o que mantém os tempos computacionais gerenciáveis (JUAN et al., 2015). As simheurísticas são uma extensão das meta-heurísticas e podem ser aplicadas como um primeiro recurso quando se está trabalhando com um problema estocástico de grande escala (CHICA et al., 2017).

A figura 2 representa um comparativo entre os principais métodos de resolução de problemas de otimização para diferentes dimensões. Métodos exatos se sobressaem em relação a otimalidade das soluções, porém apresentam grandes limitações relativas a sua escalabilidade (capacidade de trabalhar com problemas grandes), modelagem (representar sistemas da vida real com precisão), incerteza (lidar com sistemas estocásticos) e tempo computacional. As simheurísticas apresentam bons resultados, herdando das meta-heuristicas e simulação suas melhores propriedades, assim as simheurísticas conseguem trabalhar com dados estocásticos utilizando meta-heurísticas e entregar bons resultados , aplicando simulações, em baixo tempo computacional (CHICA et al., 2017).

Figura 2 – Comparação entre as abordagens de otimização baseadas em seus desempenhos relativos a diferentes dimensões.



Fonte: Adaptado de (CHICA et al., 2017).

2.5 Algoritmos genéticos

De acordo com Goldbarg, Goldbarg e Luna (2016, p. 102), "a mimetização biológica que suporta a metáfora dos AGs está associada à reprodução multicelular sexuada". Espécies que se reproduzem de forma sexuada unem seus materiais genéticos e sua prole herda genes de cada um dos seus pais, com o filho que herda as melhores características de seus pais tendo maior probabilidade de sobreviver até a fase adulta e propagar suas características (HILLIER; LIEBERMAN, 2013). Em um AG as soluções sucessoras são criadas a partir da combinação de duas soluções pais, ao invés de serem a modificação de uma única outra (RUSSELL; NORVIG, 2013).

Os AG foram apresentados por Holland (1975), desde então são uma das meta-heurísticas mais aplicadas para problemas de otimização (GOLDBARG; GOLDBARG; LUNA, 2016).

Modelos evolucionários empregam os três princípios de Darwin variação genética, hereditariedade e seleção natural (GOLDBARG; GOLDBARG; LUNA, 2016).

Segundo Goldbarg, Goldbarg e Luna (2016) a computação evolucionária é caracterizada por algumas condições, são elas:

- Sua realização depende de um processo iterativo.

- É baseada em uma população.
- Aplica o princípio darwiniano da seleção natural de acúmulos de variações genéticas.
- Variação genética: pode ocorrer por meio de mutações dos genes de um cromossomo ou como resultado de um cruzamento. Hillier e Lieberman (2013) acrescentam que mutações, por vezes, não tem efeitos ou pode ocasionar desvantagens, porém podem fornecer algumas melhorias desejáveis em uma solução.
- Hereditariedade: a população de cromossomos armazena as alterações e as transmitem para seus filhos.
- Seleção natural: representa a pressão de seleção.

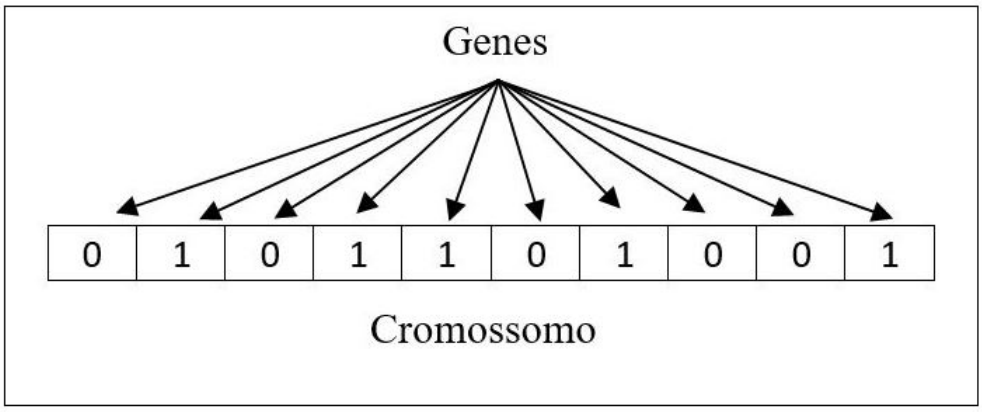
No AG, um conjunto de soluções representa uma população, cada solução é representada por um cromossomo que é composto por uma sequência de genes (YU; SEMERANO; MATTA, 2018). Um AG possui algumas estratégias para evoluir a população e gerar descendentes (YU; SEMERANO; MATTA, 2018). Em primeiro lugar devem ser selecionados os indivíduos pais, existem algumas estratégias para essa etapa como a seleção aleatória, baseada na dominância entre as soluções, método da roleta viciada e etc. Segundo Yu, Semerano e Matta (2018), o melhor cromossomo deve ser o selecionado, porém para aumentar a variabilidade das soluções indivíduos menos aptos também são aceitos. Duas soluções são escolhidos de forma aleatória para que seja feita a reprodução, com as melhores soluções tendo uma maior probabilidade de serem escolhidas (RUSSELL; NORVIG, 2013).

A figura 3 apresenta a representação de um cromossomo (uma solução) para um problema binário. A a figura 4 representa o processo de cruzamento, onde dois pais (cromossomos) são selecionados de forma aleatória. Posteriormente acontece a mistura de seus materiais genéticos por meio de um processo de cruzamento, que no presente caso consiste em um cruzamento por meio da divisão em duas partes dos cromossomos pais, resultando na geração de um filho (outra solução).

Após a seleção dos indivíduos pais, um processo de cruzamento é realizado. Segundo Hillier e Lieberman (2013), a evolução das populações se dá através da mistura de material genético entre os indivíduos. Ainda segundo Hillier e Lieberman (2013), outro fator que pode influenciar na evolução de uma população de indivíduos é taxa de mutação dos genes, sendo que esta ocorre de forma aleatória e por vezes não influencia ou piora as soluções, mas em determinados momentos conseguem melhorias significativas. A mutação pode retirar o algoritmo de ótimos locais. Por fim, após a mutação uma geração é criada e até que um critério de parada seja atendido são criadas novas gerações.

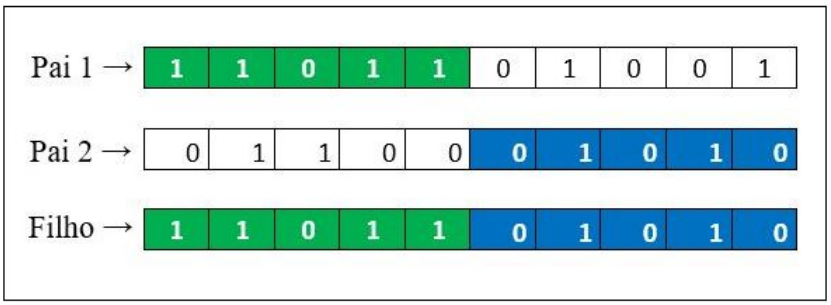
A figura 5 apresenta o pseudocódigo do AG, que recebe no seu construtor o número total de gerações e a taxa de mutação. A primeira etapa consiste na criação de uma população inicial,

Figura 3 – Cromossomo



Fonte: Autor

Figura 4 – Cruzamento



Fonte: Autor.

normalmente de forma aleatória. Posteriormente é iniciado um contador com valor zero, este será responsável por armazenar qual a geração atual. Um laço *while* é executado enquanto o número de gerações for menor que o valor do contador, dentro do laço de repetição é calculada a adequação dos cromossomos, selecionados os pais para o cruzamento, também é efetuada a mutação, por fim acontece a renovação da população de acordo com os novos indivíduos e o contador é incrementado.

2.6 Algoritmo NSGA-II

O algoritmo NSGA-II foi desenvolvido por Deb et al. (2002), e consiste em uma versão melhorada do NSGA desenvolvido por Deb e Srinivas (1995), executando um ordenamento elitista por não dominância da população, aliviando alguns dos problemas enfrentados por algoritmos que utilizam classificação e compartilhamento não dominados, como a complexidade computacional, utilizando uma abordagem elitista e não necessitando utilizar um parâmetro de compartilhamento (DEB et al., 2002). O NSGA-II é amplamente utilizado para problemas multiobjetivos, conseguindo encontrar boas soluções com baixo tempo computacional.

Figura 5 – Algoritmo Genético

```
procedimento algoritmo_genetico (num_geracoes, taxa_mutacao)
1  criação população inicial
2  contador = 0
3  enquanto num_geracoes < contador faça:
4      calcular adequação dos cromossomos
5      selecionar pais
6      cruzamento
7      efetuar mutação (taxa_mutacao)
8      renovar população
9      contador = contador + 1
10 fim enquanto;
fim algoritmo_genetico
```

Fonte: Adaptado de (GOLDBARG; GOLDBARG; LUNA, 2016)

Verma, Pant e Snasel (2021) realizaram uma extensa revisão a respeito do NSGA-II, em sua pesquisa encontraram 548 artigos que utilizam o NSGA-II para o problema do caixeiro viajante (PCV), 509 para o problema da mochila, 899 para o problema de roteamento de veículos, 2804 para o problema de alocação, 3389 para o problema de atribuição e 3688 para problemas de programação.

Segundo Deb et al. (2002) e Verma, Pant e Snasel (2021) o NSGA-II é baseado em quatro princípios principais.

1. Ordenação rápida não dominada (*fast-non-dominated sorting*): nesta etapa é executa a ordenação da população onde cada indivíduo recebe um valor para o seu *rank* de acordo com o conceito de otimalidade de pareto.

2. Operador de preservação: garante que as soluções ótimas de uma geração sejam preservadas, trasnferindo-as para a próxima geração.

3. Distância de multidão (*crowding distance*): A distância de multidão é calculada para estimar a densidade de soluções em torno de uma solução específica.

4. Seleção de aglomeração: seleciona os membros da próxima geração de acordo com os valores de seu *rank* e distância de multidão.

A seguir está detalhado o funcionamento do NSGA-II, conforme descrito por Deb et al. (2002) e Verma, Pant e Snasel (2021).

O método começa com as entradas básicas, sendo elas o tamanho da população *N*, o número máximo de iterações e a taxa de mutação, apos é realizada a criação de uma população de indivíduos P_0 de forma aleatória, que deve ter tamanho *N*. A nova população Q_0 é criada a partir da população P_0 e também tem tamanho *N*. Os índices de P_0 e Q_0 representam as gerações das populações.

Após a criação das populações estas são comparadas por meio do conceito de dominância de Pareto, a partir do nível de dominância de cada uma das soluções elas recebem um *rank*, este processo é chamado de Ordenação Rápida Não Dominada (ORND), a figura 6 mostra o pseudocódigo desta etapa. As soluções que recebem o *rank* 1 não são dominadas por nenhuma outra solução em pelo menos um dos objetivos, o *rank* 2 armazena as soluções dominadas pelas soluções de *rank* 1 e as que dominam as soluções de *rank* 3 e assim por diante até que todas as soluções estejam classificadas, a figura 7 ilustra o ORND.

De forma paralela a determinação do *rank* é calculado para cada uma das soluções o Distância de Multidão (DM) que é definido a partir da fórmula 2.15. Para calcular o DM a população deve estar ordenada de acordo com os valores de cada objetivo em ordem crescente (DEB et al., 2002).

O pseudocódigo da figura 8 ilustra esse processo do CD. O método recebe uma lista de indivíduos (fronteira pareto) e determina inicialmente para cada solução o valor zero para o DM. Posteriormente acontece a ordenação das soluções por objetivo, e os extremos, valores máximos e mínimos para cada um dos objetivos de cada uma das soluções recebe o valor de infinito, para as outras soluções é calculado o valor do CD apresentado pela fórmula 2.15.

$$crowding - distance = \sum_{i=1}^{\mathrm{k}}((f_j^{i+1} - f_j^{i-1})/(f_j^{max} - f_j^{min})) \tag{2.15}$$

Para realizar a seleção dos pais no processo de cruzamento é utilizado um operador denominado *crowded*. O *crowded* compara as soluções de acordo com seu *rank*, quando ocorrem empates uma análise dos valores do DM das soluções é utilizado como método para desempate, assim dois pontos são comparados segundo Deb et al. (2002) e Verma, Pant e Snasel (2021).

(a) Caso o *rank* das soluções selecionadas sejam diferentes, a que possuir o melhor é selecionada.

(b) Caso ambas as soluções possuem o mesmo *rank*, o valor do DM é utilizado e a que possuir o melhor valor é selecionada.

O DM estima a densidade de soluções ao redor de uma solução específica (VERMA; PANT; SNASEL, 2021).

Figura 6 – Ordenação rápida não dominada.

procedimento $fast\_non\_dominated\_sorting(P)$;

```
1  para cada p ∈ P faça:
2      S_p = 0
3      n_p = 0
4      para cada q ∈ P faça:
5          se (p < q) verdadeiro faça:
6              S_p = S_p ∪ {q}
7          senão se (q < p) verdadeiro:
8              n_p = n_p + 1
9      fim para cada;
10     se n_p = 0 verdadeiro:
11         p_rank = 1
12         F_1 = F_1 ∪ {p}
13 fim para cada;
14 i = 1
15 enquanto F_i ! = 0 faça:
16     Q = 0
17     para cada q ∈ F_i faça:
18         para cada q ∈ S_p faça:
19             n_q = n_q − 1
20             se n_q = 0 verdadeiro:
21                 Q_rank = i + 1
22                 Q = Q ∪ {q}
23         fim para cada;
24     fim para cada;
25     i = i + 1
26     F_i = Q
27 fim enquanto;
```

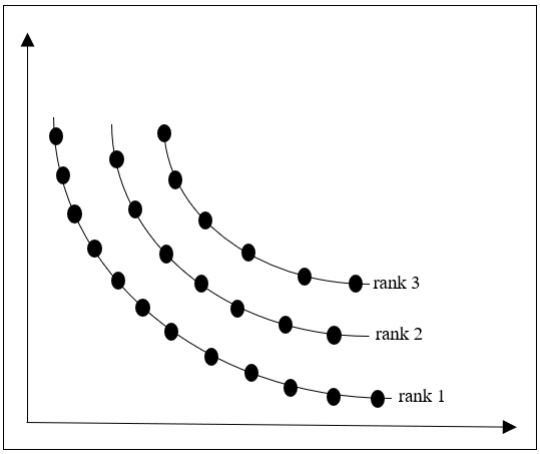
fim $fast\_non\_dominated\_sorting$

Fonte: Adaptado de (DEB et al., 2002)

A partir disso as novas populações são criadas e as novas soluções são comparadas com as soluções já rankeadas, caso alguma das novas soluções domine outra dentro do mesmo *rank* está tem seu *rank* reduzido.

O laço principal começa após a criação e ordenação da população inicial P_0, inicialmente

Figura 7 – Representação fronteiras utilizando o ORND



Fonte: Adaptado de (VERMA; PANT; SNASEL, 2021)

Figura 8 – Crowding-distance.

procedimento *crowding_distance*(I);

1 *para cada* $i \in I$ *faça*:
2 $I[\mathrm{i}] = 0$
3 *fim para cada*;
4 *para cada objetivo m faça*:
5 $I = sort(I, m)$
6 $I[1] = I[length(I)] = \infty$
7 *para cada* $i = 2$ *até* $\big((tamanho\ I) - 1\big)$*faça*:
8 $I[i] = I[i] + \frac{(I[i+1]*m - I[i-1]*m)}{f_m^{max} - f_m^{min}}$
9 *fim para cada*;
10 *fim para cada*;

fim *crowding_distance*;

Fonte: Adaptado de (DEB et al., 2002)

as populações criadas,P_i e Q_i são mescladas, está é criada a partir da primeira utilizando os processos de cruzamento e mutação. O elitismo é garantido com a mesclagem e classificação das populações (DEB et al., 2002). A figura 9 apresenta o pseudocódigo do laço principal (*main loop*) do NSGA-II.

Figura 9 – laço principal.

procedimento $main\_loop(num\_geracoes)$;

1 $R_t = P_t \; U \; Q_t$
2 $F = fast\_non\_dominated\_sorting(R_t)$
3 $P_{t+1} = 0$
4 $i = 1$
5 $enquanto \; |P_{t+1}| + |F_i| \leq num_{geracoes} faça$:
6 $\quad P_{t+1} = P_{t+1} \; U \; F_i$
7 $\quad i = i + 1$
8 $fim \; enquanto$;
9 $sort(F_i, <_n)$
10 $P_{t+1} = P_{t+1} \; U \; F_i[1:(N - |P_{t+1}|)]$
11 $P_{t+1} = cria\_nova\_pop(P_{t+1})$
12 $t = t + 1$

fim $main\_loop$;

Fonte: Adaptado de (DEB et al., 2002)

3 Metodologia

O presente trabalho é de natureza aplicada, buscou-se avaliar as aplicações de meta-heurísticas e simheurísticas para resolução do PFSP e PFSP-MO, e aplicar uma simheuristica com o NSGA-II para resolução de instâncias adaptadas por Vieira (2022) de Taillard (1993).

Tendo como abordagem o método quantitativo. O método quantitativo tem como principal característica a utilização da quantificação, seja nas modalidade de coleta de informações ou no tratamento das mesmas Pereira (2016, p. 86). Os resultados obtidos após a execução do método proposto foram avaliados por meio da comparação entre os mesmos.

Seu objetivo é descritivo, buscou se analisar de forma minuciosa o tema de estudo com pesquisas em artigos científicos e livros que o abordam, e aplicar um método para resolução do problema estudado.

A pesquisa se caracteriza como experimental, inicialmente foi determinado um objeto de estudo, sendo este o PFSP-MO, após foram definidas as instâncias a serem utilizadas, também foi definida o AG NSGA-II para resolver o problema.

4 Apresentação e discussão dos resultados

Este capítulo tem como objetivo apresentar o método utilizado para a resolução do problema de *Flow Shop* permutacional multiobjetivo e os resultados obtidos após a sua aplicação. São descritos os operadores utilizados no NSGA-II e as métricas utilizadas para calibração dos parâmetros de entrada do algoritmo.

4.1 Instâncias do problema

As instâncias utilizadas foram adaptadas de Taillard (1993) (que estão disponíveis em *http://soa.iti.es/problem-instances*) por Vieira (2022).

O procedimento de adaptação efetuado por Vieira (2022) se deu da seguinte forma, a seleção das instâncias foi realizada de forma aleatória, tendo sido escolhidas 5 instâncias de cada um dos tamanhos definidos como pequenos por Taillard (1993), que continham caracterizam por terem dimensão $n \times m$ com 10, 20, 30, 40, 50 e 60 *jobs* e 5, 10, 15 e 20 máquinas. Instâncias de dimensão de 20 e 30 *jobs* que não foram escolhidas no procedimento aleatório, elas foram adaptadas para instâncias de 15 e 25 *jobs* com 5, 10, 15 e 20 máquinas.

O conteúdo das instâncias apresenta o número de *jobs*, número de máquinas e seus respectivos tempos de processamento para cada um dos *jobs*, definido como p_{ik}. A partir dessas informações foi gerado de forma aleatória as datas de entrega utilizando a equação 4.1 (VIEIRA, 2022).

$$d_i = p_i * (1 + random * m) \tag{4.1}$$

Onde p_i representa a soma dos tempos de processamento do *job i* e *m* se refere ao número de máquinas dispostas no ambiente produtivo e random é um valor aleatório entre 0 e 1.

Instâncias maiores contendo 80 e 100 *jobs* foram geradas a partir de instâncias definidas por Taillard (1993) como grandes, estas contendo 100 *jobs* e 20, 40 e 60 máquinas.

No total foram geradas 120 instâncias por Vieira (2022).

4.2 Simheurística

Nesta seção é apresentado método simheurístico desenvolvido, a forma de representar soluções, desenvolvimento de soluções iniciais, métodos de cruzamento, mutação e calibração dos parâmetros de entrada para o método.

4.2.1 Solução inicial

A geração das soluções iniciais para o método se deu de forma aleatória, onde a probabilidade de qualquer gene ser adicionado em qualquer posição do cromossomo é igual. O tamanho da população foi determinado a partir de testes do algoritimo e análise dos resultados a partir de duas métricas: o hipervolume e a medida de cardinalidade.

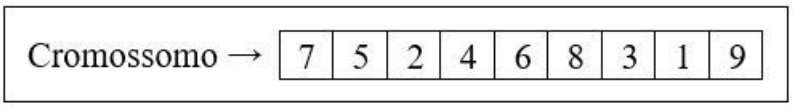
4.2.2 Representação das soluções

As soluções para o problema são representadas por meio de objetos python do tipo *Individuo*. O PFSP-MO do presente trabalho possui três objetivos cada um deles representa um atributo de um *Individuo*, o cromossomo, que também é um atributo de *Individuo*, armazena a solução, ou seja, a ordem de execução dos *n jobs* em cada uma das máquinas.

A figura 10 apresenta um cromossomo que representa uma solução aleatória para o problema. Nesta solução o primeiro *job* a ser processado é o 7, o segundo o 5 e o último o 9.

Cada *Individuo* também possui um valor de distância de multidão (*crownding distance*) como atributo bem como um rank que representa a qual fronteira a solução pertence.

Figura 10 – *Cromossomo.*



Fonte: Autor

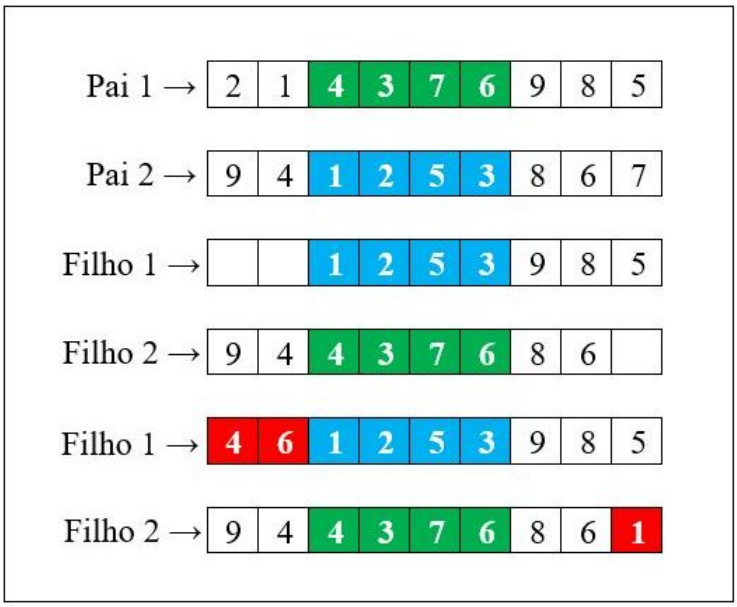
4.2.3 Cruzamento

O operador de cruzamento é o mais notável dentro os operadores que constituem um AG (TING; SU; LEE, 2010). Normalmente os AGs utilizam dois pais para realização do cruzamento, porém está não é uma regra existindo a possibilidade de utilizar mais pais para o processo de cruzamento (TING; SU; LEE, 2010). O método aplicado no presente trabalho utiliza dois pais para geração das proles.

Para realizar o cruzamento foi implementado o *partially mapped crossover* (PMX) que foi introduzido por (GOLDBERG; LINGLE, 1985).

A figura 11 apresenta o processo de cruzamento utilizado o PMX. Inicialmente de forma aleatória os pais (cromossomos) são divididos em duas partes, as proles são geradas a partir da recombinação da divisão dos pais. Quando ocorrem conflitos entre os genes, ou seja, um gene já está alocado no cromossomo por um pai e é novamente inserido por outro pai é realizado um mapeamento que remove os genes conflitantes e insere um gene ainda não adicionado ao cromossomo.

Figura 11 – *partially mapped crossover*



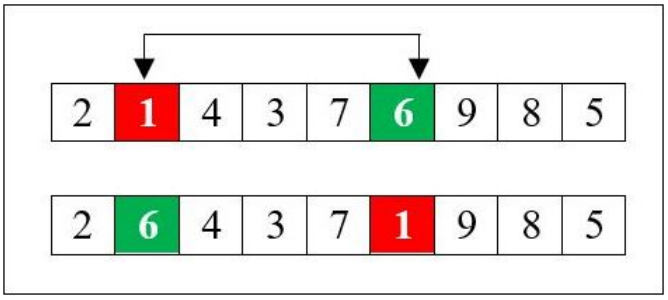
Fonte: Autor

4.2.4 Mutação

A taxa de mutação representa a probabilidade de um gene sofrer uma alteração. No caso do presente trabalho essa mutação se dá pela alteração (troca) de dois genes de posição, esta ocorrendo de forma aleatória, conforme é mostrado na Figura 12. Um baixo valor na taxa de mutação pode implicar na aproximação das soluções geradas, por outro lado valores altos podem deixar as soluções aleatórias. O processo de mutação se caracteriza como sendo um método estocástico para alteração nos genes de um cromossomo, simulando os aparatos naturais de mutação (GOLDBARG; GOLDBARG; LUNA, 2016). Segundo Katoch, Chauhan e Kumar (2020), a taxa de mutação garante que as populações mantenham diversidade genéticas.

No presente trabalho a taxa de mutação foi definida a partir de testes com as instâncias.

Figura 12 – Mutação em um gene



Fonte: Autor

4.2.5 Número de gerações

O número de gerações, assim como a taxa de mutação foi selecionado a partir de testes com as instâncias utilizadas no trabalho.

O número de gerações é responsável por determinar a quantidade de repetições do código do laço principal, definindo assim a cada nova iteração uma população. Valores altos para o número de gerações podem chegar a resultados melhores, selecionando melhores soluções para a FP, porém existe um maior consumo de tempo computacional.

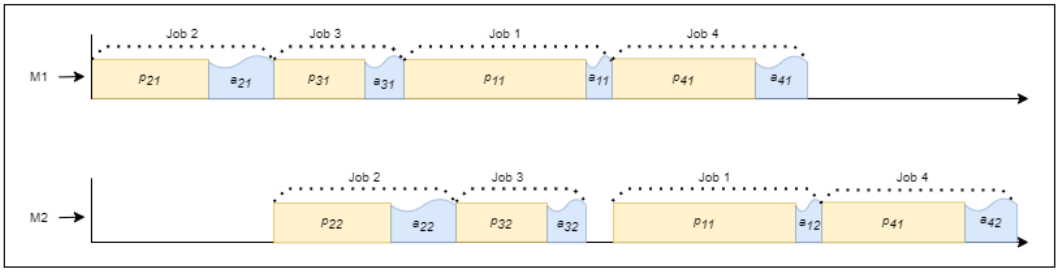
4.2.6 Parâmetro estocástico

Para simular um ambiente real de produção um parâmetro estocástico é adicionado ao problema, buscando simular possíveis atrasos, falhas de máquinas e tempos de preparação das máquinas. Cota et al. (2022) descrevem o parâmetro estocástico como a_{ij} que é gerado de forma aleatória e adicionado ao tempo de processamento p_{ij}.

O cálculo do valor do parâmetro estocástico é apresentado por Vieira (2022) e é definido pela equação 4.2, onde w é um valor selecionado aleatoriamente com distribuição normal de média 10 e desvio padrão 2 (VIEIRA, 2022). A figura 13 exibe o comportamento do parâmetro estocástico, seu valor é definido de forma conjunta ao tempo de processamento e somado ao *makespan*.

$$a_{ij} = w/100 * p_{ij} \tag{4.2}$$

Figura 13 – Representação do parâmetro estocástico a_{ik}



Fonte: Adaptado de (VIEIRA, 2022)

4.2.7 Simulação

Para cada solução acrescentada a FP ótima é executada uma simulação rápida com 30 replicações, a qual é executada dentro do corpo do algoritmo NSGA-II. Após a sua execução as soluções são comparadas por meio do conceito de dominância de pareto e as soluções dominadas são retiradas da FP ótima.

Ao término da execução do algoritmo uma grande simulação com 500 replicações é executada, e assim como a simulação rápida as soluções são analisadas e a fronteira ótima é retornada contendo apenas indivíduos não dominados.

A figura 14 apresenta o pseudo-código para a simulação aplicada. Inicialmente para cada uma das soluções que compõe a FP ótima é executado o código das linhas 1 a 8, sendo esta a etapa responsável por executar o cálculo do parâmetro estocástico, a linha 7 é responsável por calcular os valores para os objetivos do PFSP-MO. Posteriormente, as soluções são adicionadas a um vetor de soluções, e ao fim do número de replicações é chamado o método *fast_non_dominated_sorting* onde as soluções são comparadas por meio do conceito de dominância de pareto e as soluções dominadas são retiradas da FP ótima.

Figura 14 – Pseudo-código simulações.

procedimento *simulacao*(*numero_replicacoes*);

```
1  para cada s ∈ fronteira_pareto faça
2      enquanto repeticoes ≤ numero_replicacoes faça
3          (a_ik) = avaliacao_estocastica
4          sol[s] ← solucao(a_ik)
5          repeticoes += 1
6      fim enquanto
7      s = sol[s]/numero_replicacoes
8      solucoes ← s
9  fim para cada
10 fast_non_dominated_sorting(solucoes)
```

fim *simulacao*

Fonte: Adaptado de (VIEIRA, 2022)

4.3 Definição dos parâmetros iniciais do método

Para a definição dos parâmetros iniciais do método foram utilizadas duas métricas, a diferença de hipervolume e medida de cardinalidade.

- Diferença de hipervolume: o hipervolume (I_H) foi apresentado por (ZITZLER; THIELE, 1999), ele calcula o volume do espaço das soluções da fronteira pareto a partir de um ponto de referência, sendo calculado a partir da equação 4.3. Quanto maior for o valor do hipervolume melhor é a solução, com ela se aproximando mais da fronteira pareto

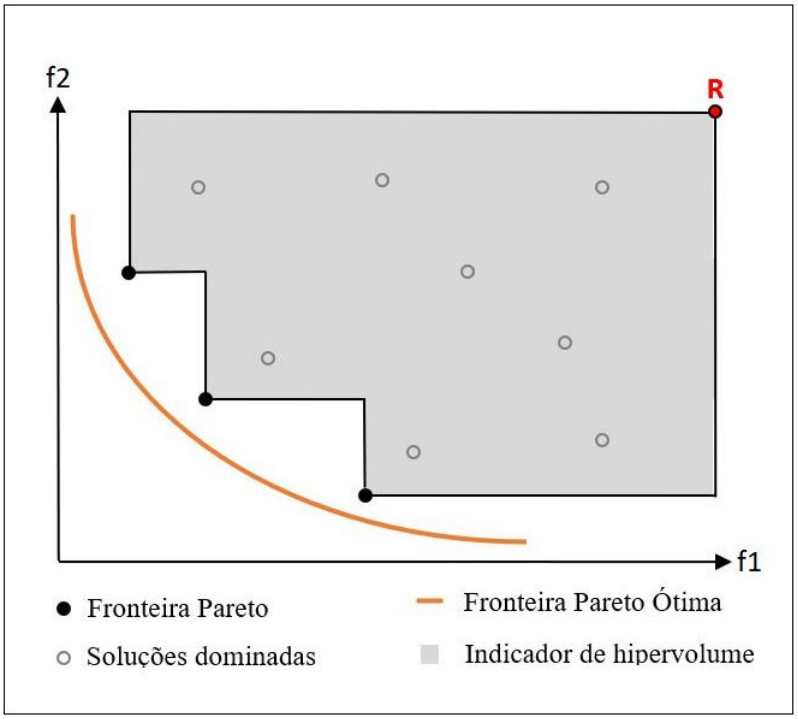
ótima (VARGAS et al., 2014). A figura 15 apresenta um exemplo para um espaço de dois objetivos, aprsentando a fronteira pareto, a fronteira pareto ótima, o hipervolume e as soluções dominadas. O ponto de referência pode ser definido de formas diferentes, sendo o mais usual o ponto extremo máximo da fronteira pareto, ou seja, o maior valor para o objetivo. Para problemas multiobjetivos, os pontos de referência são definidos de acordo com o número de objetivos, sendo para o presente trabalho foram definidos três pontos (X_1, X_2, X_3).

$$I_H(Q) = \bigcup_{1}^{|Q|} v_i \tag{4.3}$$

A diferença de hipervolume é apresentada pela equação 4.4, onde valores menores para I_H representam maior qualidade (ÁLVAREZ; RISCO-MARTíN; COLMENAR, 2016).

$$I_H^-(Q) = I_H(R) - I_H(Q) \tag{4.4}$$

Figura 15 – Representação do hipervolume, froneteira pareto, fronteira pareto ótima e soluções dominadas.



Fonte: Adaptado de (ÁLVAREZ; RISCO-MARTíN; COLMENAR, 2016).

- Medida de Cardinalidade: A medida de cardinalidade, introduzida por Zitzler et al. (2003), representa o número de soluções pertencentes a fronteira pareto de referência que fazem parte da fronteira pareto analisada. A fronteira pareto de referência utilizada no presente

Tabela 1 – Instâncias utilizadas para a calibração das métricas.

Instância	Número Jobs	Número Máquinas
VFR20_10_2_Gap	20	10
VFR30_15_4_Gap	30	15
VFR40_15_9_Gap	40	15
VFR80_20_1_Gap	80	20
VFR100_20_5_Gap	100	20

Tabela 2 – Valores utilizados para calibração dos parâmetros inicias do método.

Parâmetro	Valor 1	Valor 2	Valor 3	Valor 4	Valor 5	Valor6
Taxa Mutação	0,01	0,02	0,03	0,04	0,05	0,06
Número de gerações	100	300	500	800	1000	2000
Tamanho populacional	10	20	30	40	50	100

trabalho, foi a união entre duas fronteiras de soluções, sendo as fronteiras construídas com valores aleatórios de taxa de mutação, tamanho populacional e número de gerações.

O algoritmo proposto foi executado com 6 instâncias escolhidas aleatoriamente. A tabela 1 apresenta as instâncias selecionadas para a calibração dos parâmetros de entrada do algoritmo. Foram executadas dez vezes cada uma das instâncias com valores de taxa de mutação, número de gerações e tamanho populacional selecionados de forma aleatória, sendo essas comparadas pares a pares utilizando as métricas descritas anteriormente. A comparação em pares se deu para que se pudesse criar um conjunto referência de soluções, esse conjunto é composto pela união das duas FP. A seleção dos parâmetros para execução dos teste foi realizada a partir de um algoritmo, que selecionou de forma aleatória os valores de cada um dos três parâmetros de entrada.

O intervalo de valores para os parâmetros inicias está apresentado na tabela 2.

4.3.1 Análise resultados métricas

Após as execuções os resultados foram analisados a fim de se obter os melhores valores para os parâmetros de entrada do método.

Inicialmente foram avaliados os resultados a partir da métrica de medida de cardinalidade para definição do tamanho da população. As tabelas 3, 4, 5, 6, 7 e 8 apresentam os resultados das execuções dos testes. A coluna Mut 1 e Mut 2 armazenam os valores da taxa de mutação para cada uma das execuções da instância apresentada na coluna instância. O número de gerações está localizado na coluna Num ger 1 e Num ger 2, para cada uma das execuções. Também estão descritos o tamanho populacional das execuções de teste em Tam pop 1 e Tam pop 2. O tamanho da fronteira pareto para a execução 1 e 2 são apresentados na coluna Front 1 e Front 2. Por fim, são descritos os resultados das métricas, para a medida de cardinalidade as colunas Card 1 e 2 descrevem os valores e para o hipervolume as colunas Hiper 1 e 2.

Tabela 3 – Melhores valores para o tamanho populacional de acordo com a medida de cardinalidade

Mut 1	Mut 2	Num ger 1	Num ger 2	Tam pop 1	Tam pop 2	Instância	Front 1	Front 2	Med Card 1	Med Card 2
0,04	0,05	1000	800	***100***	10	VRF20_10_2	66	10	***55***	9
0,05	0,03	500	2000	20	***100***	VRF20_10_2	21	57	8	***48***
0,01	0,02	2000	500	***100***	20	VRF30_15_4	117	25	***77***	20
0,03	0,02	500	1000	***100***	30	VRF30_15_4	106	44	***66***	38
0,01	0,05	500	500	***50***	30	VRF40_15_9	68	37	***66***	37
0,01	0,04	500	100	***50***	20	VRF40_15_9	64	21	***63***	21
0,01	0,01	1000	1000	***100***	30	VRF80_20_1	60	35	***59***	34
0,04	0,01	500	2000	30	***100***	VRF80_20_1	30	61	5	***61***
0,02	0,04	300	100	***100***	30	VRF100_20_5	54	18	***54***	18
0,03	0,02	300	800	***100***	30	VRF100_20_5	55	33	***54***	33

Tabela 4 – Melhores valores para o tamanho populacional de acordo com o hipervolume

Mut 1	Mut 2	Num ger 1	Num ger 2	Tam pop 1	Tam pop 2	Instância	Front 1	Front 2	Hiper 1	Hiper 2
0,05	0,03	500	2000	20	***100***	VRF20_10_2	21	57	60	***530,3***
0,04	0,05	1000	800	***100***	10	VRF20_10_2	66	10	***584,7***	43,9
0,01	0,02	2000	500	***100***	20	VRF30_15_4	117	25	***3.303,2***	687,9
0,03	0,02	500	1000	***100***	30	VRF30_15_4	106	44	***2.188,4***	1.035,4
0,01	0,05	500	500	***50***	30	VRF40_15_9	68	37	***48.190,6***	27.819,1
0,05	0,01	300	1000	***100***	30	VRF40_15_9	66	40	***41.031,1***	36.427,8
0,04	0,05	300	800	30	***40***	VRF80_20_1	21	28	88.685,7	***163.893,3***
0,01	0,01	1000	1000	***100***	30	VRF80_20_1	60	35	***286.995,4***	122.749,6
0,01	0,01	100	2000	50	***40***	VRF100_20_5	23	51	***242.532,0***	711.776,6
0,02	0,04	300	100	***100***	30	VRF100_20_5	54	18	***504.970,6***	128.245,6

A tabela 3 ilustra os melhores resultados da métrica (destacados em negrito) e quais os valores para o tamanho da população (destacados em negrito) . Os melhores resultados obtidos foram para o tamanho populacional igual a 100. Como a seleção do valores para os parâmetros inicias do método foram aleatórias a instância VFR40_15_9_GAP não apresentou testes com esse valor, mas os melhores resultados para ela foram encontrados com os maiores valores para o tamanho populacional em que a instância foi executada, neste caso 50.

Utilizando a diferença de hipervolume para avaliar o melhor valor para o tamanho populacional é perceptível que os maiores valores, assim como para a medida de cardinalidade quanto maior o valor do tamanho da população melhor é o resultado da métrica. Apenas na instância VFR80_20_1_GAP isso não ocorre como apresentado na tabela 4, porém o número de gerações que será analisado posteriormente afeta significativamente esses valores.

A métrica de medida de cardinalidade mostra que normalmente valores mais altos para o número de gerações conseguem melhores resultados, com 2000 e 500 aparecendo mais vezes com resultados melhores, porém quando ambos os valores aparecem juntos em comparação 2000 gerações se sobressaem sobre 500 como pode ser observado na tabela 5. A tabela 6 apresenta em verde os melhores valores para o número de gerações. Nessa figura fica evidente também que 2000 é o melhor valor para o número de replicações, onde existe também bons resultados para 500, porém assim como na métrica anterior 2000 se sobressai.

O operador de mutação seguindo direção contrária aos outros operadores apresentou melhores resultados com valores mais baixos com 0,01 sendo o valor mais presente na amostra que contém os melhores resultados. A tabela 7 mostra que para a métrica de medida de cardinalidade,

Tabela 5 – Melhores valores para o número de gerações de acordo com a medida de cardinalidade

Mut 1	Mut 2	Num ger 1	Num ger 2	Tam pop 1	Tam pop 2	Instância	Front 1	Front 2	Med Card 1	Med Card 2
0,04	0,05	***1000***	800	100	10	VRF20_10_2	66	10	***55***	9
0,05	0,03	500	***2000***	20	100	VRF20_10_2	21	57	8	***48***
0,01	0,02	***2000***	500	100	20	VRF30_15_4	117	25	***77***	20
0,03	0,02	***500***	1000	100	30	VRF30_15_4	106	44	***66***	38
0,01	0,05	***500***	500	50	30	VRF40_15_9	68	37	***66***	37
0,01	0,04	***500***	100	50	20	VRF40_15_9	64	21	***63***	21
0,01	0,01	***1000***	1000	100	30	VRF80_20_1	60	35	***59***	34
0,04	0,01	500	***2000***	30	100	VRF80_20_1	30	61	5	***61***
0,02	0,04	***300***	100	100	30	VRF100_20_5	54	18	***54***	18
0,03	0,02	***300***	800	100	30	VRF100_20_5	55	33	***54***	33

Tabela 6 – Melhores valores para o número de gerações de acordo com o hipervolume

Mut 1	Mut 2	Num ger 1	Num ger 2	Tam pop 1	Tam pop 2	Instância	Front 1	Front 2	Hiper 1	Hiper 2
0,05	0,03	500	***2000***	20	100	VRF20_10_2	21	57	60	***530,3***
0,04	0,05	***1000***	800	100	10	VRF20_10_2	66	10	***584,7***	43,9
0,01	0,02	***2000***	500	100	20	VRF30_15_4	117	25	***3.303,2***	687,9
0,03	0,02	***500***	1000	100	30	VRF30_15_4	106	44	***2.188,4***	1.035,4
0,01	0,05	***500***	500	50	30	VRF40_15_9	68	37	***48.190,6***	27.819,1
0,05	0,01	***300***	1000	100	30	VRF40_15_9	66	40	***41.031,1***	36.427,8
0,04	0,05	300	***800***	30	40	VRF80_20_1	21	28	88.685,7	***163.893,3***
0,01	0,01	***1000***	1000	100	30	VRF80_20_1	60	35	***286.995,4***	122.749,6
0,01	0,01	100	***2000***	50	40	VRF100_20_5	23	51	***242.532,0***	711.776,6
0,02	0,04	***300***	100	100	30	VRF100_20_5	54	18	***504.970,6***	128.245,6

Tabela 7 – Melhores valores para o operador de mutação de acordo com a medida de cardinalidade

Mut 1	Mut 2	Num ger 1	Num ger 2	Tam pop 1	Tam pop 2	Instância	Front 1	Front 2	Med Card 1	Med Card 2
0,04	0,05	1000	800	100	10	VRF20_10_2	66	10	***55***	9
0,05	***0,03***	500	2000	20	100	VRF20_10_2	21	57	8	***48***
0,01	0,02	2000	500	100	20	VRF30_15_4	117	25	***77***	20
0,03	0,02	500	1000	100	30	VRF30_15_4	106	44	***66***	38
0,01	0,05	500	500	50	30	VRF40_15_9	68	37	***66***	37
0,01	0,04	500	100	50	20	VRF40_15_9	64	21	***63***	21
0,01	0,01	1000	1000	100	30	VRF80_20_1	60	35	***59***	34
0,04	***0,01***	500	2000	30	100	VRF80_20_1	30	61	5	***61***
0,02	0,04	300	100	100	30	VRF100_20_5	54	18	***54***	18
0,03	0,02	300	800	100	30	VRF100_20_5	55	33	***54***	33

das dez execuções que apresentaram os melhores resultados e estão apresentadas na figura, cinco utilizaram como o operador de mutação o valor 0,01.

A métrica de diferença de hipervolume apresenta resultados parecidos, mesmo sendo um pouco mais diversa, 40% da amostra apresenta os melhores resultados com a taxa de mutação sendo 0,01 como pode ser observado na tabela 8.

Após as análises os valores selecionados para os parâmetros de entradas do método são apresentados na tabela 9.

4.4 Resultados obtidos

Os resultados apresentados a seguir foram obtidos após a execução da simheurística utilizando as 120 instâncias descritas no início desta seção. A execução foi realizada em um computador com processador Ryzen 7 1800x 3,6GHz, 64GB de memória RAM e sistema

Tabela 8 – Melhores valores para o operador de mutação de acordo com a diferença de hipervolume

Mut 1	Mut 2	Num ger 1	Num ger 2	Tam pop 1	Tam pop 2	Instância	Front 1	Front 2	Hiper 1	Hiper 2
0,05	***0,03***	500	2000	20	100	VRF20_10_2	21	57	60	***530,3***
0,04	0,05	1000	800	100	10	VRF20_10_2	66	10	***584,7***	43,9
0,01	0,02	2000	500	100	20	VRF30_15_4	117	25	***3.303,2***	687,9
0,03	0,02	500	1000	100	30	VRF30_15_4	106	44	***2.188,4***	1.035,4
0,01	0,05	500	500	50	30	VRF40_15_9	68	37	***48.190,6***	27.819,1
0,05	0,01	300	1000	100	30	VRF40_15_9	66	40	***41.031,1***	36.427,8
0,04	***0,05***	300	800	30	40	VRF80_20_1	21	28	88.685,7	***163.893,3***
0,01	0,01	1000	1000	100	30	VRF80_20_1	60	35	***286.995,4***	122.749,6
0,01	***0,01***	100	2000	50	40	VRF100_20_5	23	51	***242.532,0***	711.776,6
0,02	0,04	300	100	100	30	VRF100_20_5	54	18	***504.970,6***	128.245,6

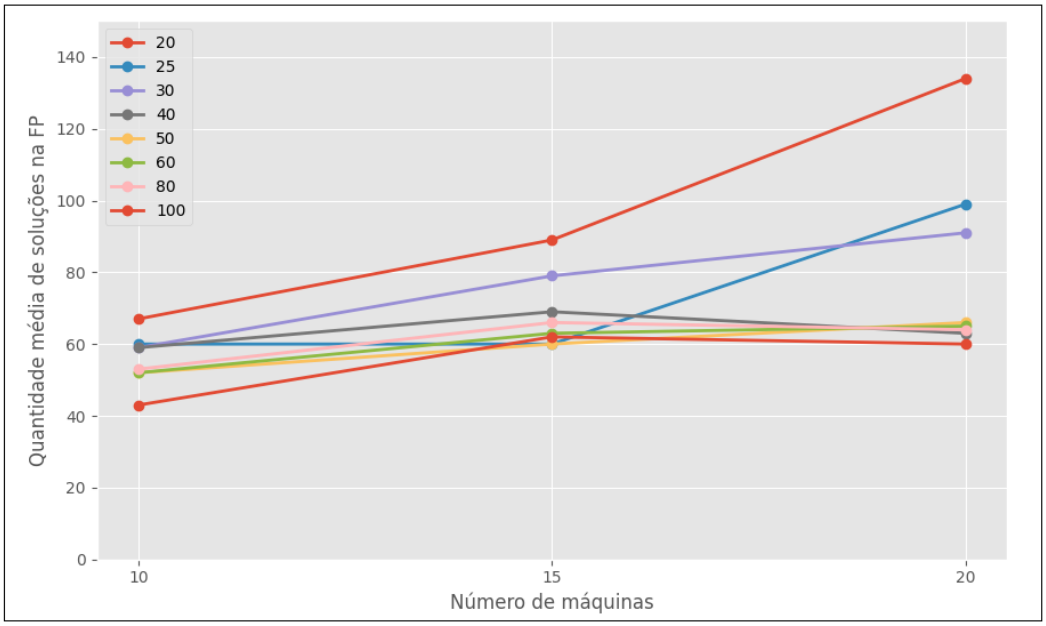
Tabela 9 – Valores selecionados para os parâmetros inicias do método.

Parâmetro	Valor
Taxa Mutação	0,01
Número de gerações	2000
Tamanho populacional	100

operacional Windows 10. Cada instância foi executada por 3600 segundos, quando o número máximo de gerações foi atingido uma nova rodada era iniciada.

A figura 16 apresenta o número médio de soluções por número de *jobs* na FP de acordo com o número de máquinas. Quanto maior o número de máquinas, nota-se um aumento da quantidade média de soluções na FP, também pode-se perceber que a instância com menor número de *jobs* apresenta a maior quantidade média de soluções.

Figura 16 – Média de soluções (baseado no número de *jobs*) na FP por número de máquinas



Fonte: Autor.

Observando a tabela 10 fica nítido que instâncias com maior número de máquinas, salvo

Tabela 10 – Quantidade média de soluções na FP por instância.

Instância	Quantidade média de soluções
20 10	67
20 15	89
20 20	134
25 10	60
25 15	60
25 20	99
30 10	59
30 15	79
30 20	91
40 10	59
40 15	69
40 20	63
50 10	52
50 15	60
50 20	66
60 10	52
60 15	63
60 20	65
80 10	53
80 15	66
80 20	64
100 10	43
100 15	62
100 20	60

exceções, apresentam uma maior quantidade de soluções na FP, a média das instâncias com 20 *jobs* e 20 máquinas apresentou 134 soluções na FP, enquanto a de 30 *jobs* e 20 máquinas apresentou 91 soluções.

Caso o tomador de decisão que avalie os resultados busque como objetivo principal a redução de um dos objetivos em específico pode ser selecionado o de sua preferência. A tabela 11 apresenta alguns indivíduos com o cromossomo e os valores dos objetivos selecionados da FP de forma aleatória. Caso o *makespan* seja o objetivo prioritário buscado na minimização, a solução de id 1 pode ser escolhida pois apresenta o menor valor para este objetivo. Quando o atraso total é a prioridade na escolha, a solução de id 3 pode ser escolhida apresentando um valor de 0 unidades de tempo, menor valor para o atraso sendo um ótimo global para este objetivo. Por fim, se a antecipação total for escolhida como prioritária, a solução de id 6 deverá ser selecionada com um valor de 29.366 unidades de tempo. Caso a opção seja um melhor valor para os três objetivos somados a solução de id 6 apresenta o melhor valor 32.257, porém apresenta um *makespan* mais alto que a média das soluções apresentadas e também um alto atraso total, o segundo maior da tabela.

As variações das soluções permitem que seja escolhida uma solução que melhor se adeque

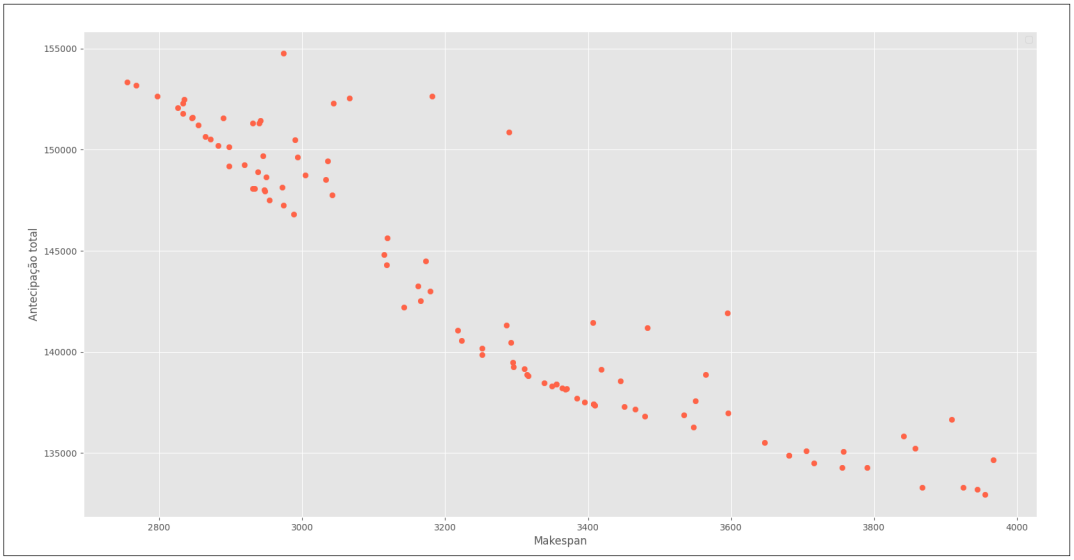
Tabela 11 – Resultados da instância 20_10_2.

id	Cromossomo	Makespan	Atraso	Antecipação
1	[17, 14, 3, 12, 4, 11, 10, 13, 19, 7, 5, 1, 16, 18, 8, 6, 9, 15, 20, 2]	1.737	98	36.048
2	[17, 14, 3, 12, 4, 11, 5, 13, 7, 1, 10, 19, 16, 8, 18, 6, 9, 15, 20, 2]	1.718	115	36.058
3	[4, 17, 13, 2, 6, 19, 8, 1, 10, 7, 15, 14, 9, 5, 20, 18, 12, 11, 3, 16]	2.579	0	31.104
4	[6, 17, 12, 11, 5, 13, 19, 4, 10, 7, 16, 8, 18, 1, 3, 14, 15, 20, 9, 2]	1.846	621	33.619
5	[17, 14, 3, 12, 11, 5, 13, 7, 19, 4, 1, 10, 16, 8, 18, 6, 9, 15, 20, 2]	1.751	345	35.917
6	[6, 17, 9, 5, 4, 13, 14, 18, 1, 12, 19, 11, 8, 3, 16, 10, 7, 15, 2, 20]	2.287	604	29.366

ao momento específico para a escolha, prazos maiores ou menores para um determinado produto final, ou a opção por ponderar um objetivo específico por exemplo quando o atraso deve ser minimizado.

A figura 18 apresenta a distribuição das soluções em um gráfico 2D considerando o *makespan* e a antecipação total, é notável que a redução do valor de um dos objetivos aumenta o valor de outro confirmando que os objetivos são conflitantes, quando o *makespan* tem valores menores a antecipação apresenta valores maiores. O melhor valor para o *makespan* na figura 18 apresenta um dos maiores valores para a antecipação total. De forma análoga o melhor valor para a antecipação total representa o segundo maior valor para o *makespan*.

Figura 17 – Espaço de soluções instância - 30_15_8 - *Makespan* x Atraso total

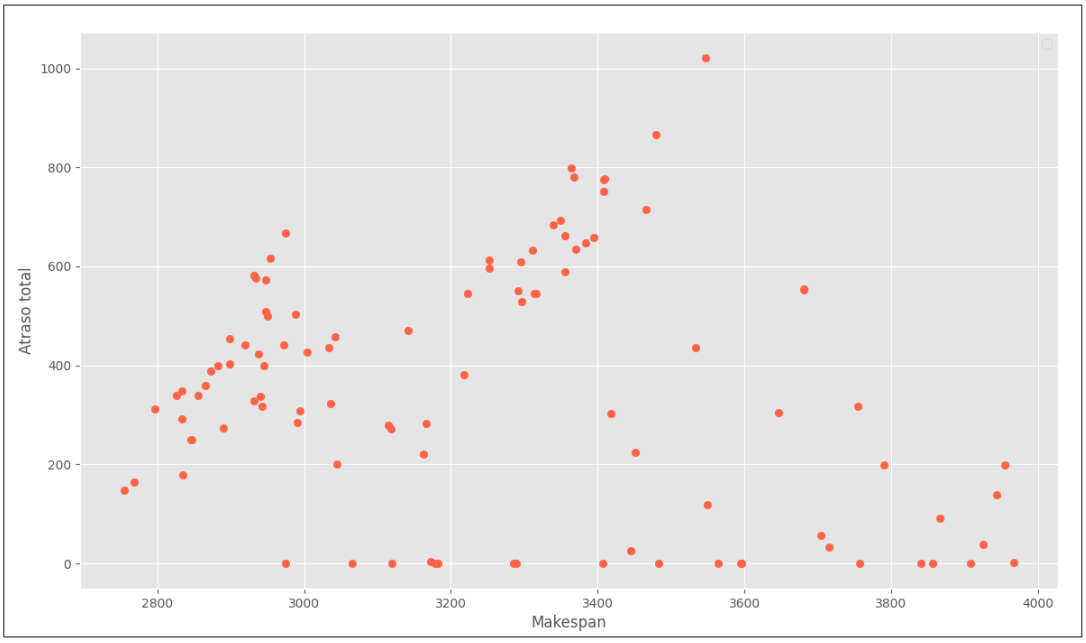


Fonte: Autor.

Analisando a figura 17 podemos notar um resultado diferente do observado na 18, aqui valores ótimos de atraso total são encontrados para valores com baixo ou alto *makespan*, não sendo tão perceptível a conflitância entre os objetivos.

Um ponto notável é que as soluções apresentam variações significativas entre os valores de seus objetivos, a tabela 12 apresenta algumas soluções da FP ótima para a instância 50 10, onde pode-se perceber grandes variações dos valores dos três objetivos, o que mostra a gama de opções

Figura 18 – Espaço de soluções instância - 30_15_8 - *Makespan* x Antecipação total



Fonte: Autor.

Tabela 12 – Valores dos objetivos de soluções da instância 50 10

id	Makespan	Atraso total	Antecipação total
1	4.875	11.497	9.556
2	4.051	3.587	22.270
3	3.663	12.356	37.121
4	4.948	12.787	9.142
5	3.514	19.803	43.820

disponíveis para o tomador de decisões. O atraso total por exemplo sofre variações de 3.587 unidades até 19.803, bem como o *makespan* que variou dentro das soluções, que foram obtidas de forma aleatória, entre 3.514 unidades até 4.948, porém entregando melhoras significativas em outros objetivos. Outro ponto de destaque é a solução de id 5, que apresentou o melhor valor para o makespan, porém obteve os piores valores para o atraso total e a antecipação total.

5 Conclusões e considerações finais

Neste trabalho foi apresentado e discutido o PFSP-MO estocástico, problema que é comum dentro da indústria, no qual buscou se minimizar três objetivos de forma simultânea o *makespan*, atraso total e antecipação total. O PFSP se caracteriza como um problema onde *n jobs* devem ser executados em *m* e a ordem de execução é a mesma em todas as máquinas.

Buscou-se a implementação de uma simheurística para resolução do problema, sendo um AG utilizado para o refinamento das soluções. O AG selecionado foi o NSGA-II, por sua capacidade já comprovada de resolver problemas multi-objetivos (MO) com bom tempo computacional e entregando soluções de boa qualidade. O método de otimalidade de Pareto foi utilizado para verificar a qualidade das soluções e compará-las.

O método foi executado utilizando 120 instâncias adaptadas por Vieira (2022) de Taillard (1993), sendo estas com dimensões *n x m* com a menor contendo 20 *jobs* e 10 máquinas e a maior com 100 *jobs* e 20 máquinas.

Aplicaram-se duas métricas para avaliação da qualidade das soluções utilizadas para a calibração dos parâmetros de entrada do AG, hipervolume e medida de cardinalidade. Os resultados das métricas para cada instância foram comparados buscando os melhores valores para cada um dos parâmetro de entrada.

O NSGA-II se mostrou um bom método para resolução do problema, conseguindo boas soluções em um bom tempo computacional e apresentou um bom resultado junto a simulação com soluções diversas que conseguem entregar bons valores para cada um dos objetivos, mesmo com instâncias maiores.

Os resultados obtidos e apresentados no presente trabalho contribuem para um método que pode gerar boas soluções para problemas estocásticos e que visam otimizar objetivos conflitantes, problemas estes que são comuns dentro do cotidiano de diversos setores da indústria.

Como trabalhos futuros, aconselha-se a aplicação do método para outros ambientes de produção, como alguns apresentados dentro do presente trabalho como, por exemplo, o problema de *job shop*. Também a aplicação de outra meta-heurística para resolver o problema, como por exemplo, uma meta-heurística colônia de formigas.

Referências

ABDEL-BASSET, M.; MANOGARAN, G.; EL-SHAHAT, D. A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem. *ELSEVIER*, Egypt, v. 85, p. 129–145, 2018. 1, 7, 8

ARENALES, M. et al. *Pesquisa Operacional: para cursos de enegnharia*. Rio de Janeiro: Elsevier, 2015. 1, 4, 6, 7, 9

CARDOSO, W. *Planejamento e Controle da Produção (PCP): a teoria na prática*. São Paulo: Edgard Blücher Ltda, 2021. 1

CHIAVENATO, I. *Planejamento e controle da produção*. Barueri: Manole, 2008. 1

CHICA, M. et al. Why simheuristics? benefits, limitations, and best practices when combining metaheuristics with simulation. *Operations Research Perspectives*, Available at SSRN: https://ssrn.com/abstract=2919208, 2017. 10, 11

COTA, F. dos R. et al. An simheuristic approach to solve the multi-objective permutation flowshop problem. *Brazilian Journal of Development*, Brasil, v. 8, n. 1, p. 5550–5563, 2022. 5, 10, 23

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, v. 6, n. 2, p. 182–197, 2002. 3, 13, 14, 15, 16, 17, 18

DEB, K.; SRINIVAS, N. Muiltiobj ective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, Massachusetts, v. 2, n. 3, p. 221–248, 1995. 13

EDDALY, M.; JARBOUI, B.; SIARRY, P. Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. *Journal of Computational Design and Engineering*, Paris, v. 3, n. 4, p. 295–311, 2016. 8

FERONE, D. et al. A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem. *INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH*, p. 1–24, 2019. 8

GAREY, M. R.; JOHNSON, D. S. Two-processorschedulingwithstart-timesanddeadlines. *SIAMJ.COMPUT.*, v. 6, n. 3, p. 416–426, 1977. 2

GOLDBARG, M. C.; GOLDBARG, E. G.; LUNA, H. P. L. *Otimização combinatória e meta-heurísticas*. Rio de Janeiro: LTC, 2016. 2, 3, 4, 11, 14, 22

GOLDBERG, D. E.; LINGLE, R. Alleles, loci, and the traveling salesman problem. *Lawrence Erlbaum Associates*, p. 154–159, 1985. 21

GRIMME, C.; LEPPING, J. Multi-criteria scheduling: an agent-based approach for expert knowledge integration. *Springer Science+Business Media*, Dortmund, 2011. 6

HILLIER, F. S.; LIEBERMAN, G. J. *Introdução a Pesquisa Operacional*. Porto Alegre: AMGH, 2013. 2, 4, 11, 12

HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Michigan: U Michigan Press, 1975. 2, 11

HUANG, K.-W. et al. A hybrid crow search algorithm for solving permutation flow shop scheduling problems. *Applied Sciences*, v. 9, n. 7, 2019. 8

JUAN, A. A. et al. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, v. 2, n. 1, p. 62–72, 2015. 10

KATOCH, S.; CHAUHAN, S. S.; KUMAR, V. A review on genetic algorithm: past, present, and future. *Springer Science+Business Media*, p. 117–132, 2020. 22

LAGE, M. *Planejamento e Controle da Produção teoria e prática*. Rio de Janeiro: LTC, 2019. 1, 6

LI, X.; MA, S. Multi-objective memetic search algorithm for multi-objective permutation flow shop scheduling problem. *IEEAccess*, Changchun, v. 4, p. 2154–2165, 2016. 1, 2, 5, 8

NOGUEIRA, J. P. de C. M. et al. Hybrid grasp heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties. *Electronic Notes in Theoretical Computer Science*, p. 53–72, 2014. 6

PEREIRA, J. M. *Manual de Metodologia da Pesquisa Científica*. São Paulo: Atlas, 2016. 19

PINEDO, M. L. *Scheduling Theory, Algotithms, and Systems*. New York: Springer, 2012. 1, 7

RAHMAN, H. F.; SARKER, R.; ESSAM, D. A genetic algorithm for permutation flow shop scheduling under make to stock production system. *Computers e Industrial Engineering*, Australia, p. 12–24, 2015. 8

RIFAI, A. P. et al. Multi-operator hybrid genetic algorithm-simulated annealing for reentrant permutation flow-shop scheduling. *ASEAN Engineering Journal*, v. 11, n. 3, p. 109–126, 2020. 2, 6, 8

RUIZ, R.; STüTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, p. 2033–2049, 2006. 7

RUSSELL, S.; NORVIG, P. *Inteligência Artificial*. Rio de Janeiro: Elsevier Editora LTDA., 2013. 11, 12

TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, Lausanne, n. 64, p. 278–285, 1993. 3, 19, 20, 33

TING, C.-K.; SU, C.-H.; LEE, C.-N. Multi-parent extension of partially mapped crossover for combinatorial optimization problems. *Expert Systems with Applications*, China, v. 37, p. 1879–1886, 2010. 21

VARGAS, D. E. C. et al. AnÁlise do desempenho de algoritmos baseados em evoluÇÃo diferencial acoplados a uma tÉcnica de penalizaÇÃo adaptativa em problemas de otimizaÇÃo estrutural multiobjetivo com restriÇÕes. *SIMMEC / EMMCOMP 2014*, Juiz de Fora, 2014. 25

VERMA, S.; PANT, M.; SNASEL, V. A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems. Ostrava, v. 9, p. 57757–57791, 2021. 14, 15, 17

VIEIRA, N. H. *ABORDAGENS SIMHEURÍSTICAS PARA O PROBLEMA DE FLOW SHOP PERMUTACIONAL MULTIOBJETIVO*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Ouro Preto, 2022. 3, 19, 20, 23, 24, 33

XIE, Z. et al. An effective hybrid teaching?learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software*, China, v. 77, p. 35–47, 2014. 8

YU, C.; SEMERANO, Q.; MATTA, A. A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Computers Operations Research*, Italy, v. 1006, p. 211–219, 2018. 12

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, v. 3, n. 4, p. 257–271, 1999. 24

ZITZLER, E. et al. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, v. 7, n. 2, p. 117–132, 2003. 25

ÁLVAREZ, J. D.; RISCO-MARTíN, J. L.; COLMENAR, J. M. Multi-objective optimization of energy consumption and execution time in a single level cache memory for embedded systems. *The Journal of Systems and Software*, Spain, v. 111, p. 200–212, 2016. 25