**Universidade Federal de São Carlos**

**Pró-Reitoria de Pesquisa**

**Coordenadoria de Iniciação Científica e Tecnológica**

# Relatório final PIBIC

**Título da pesquisa:**

**Análise dos operadores de crossover em algoritmos genéticos aplicados a problemas de Flow Shop Scheduling**

**Nome do aluno:**

Lucas Gabriel Malheiros Machado Silva

**Nome do orientador:**

Prof. Dr. Roberto Fernandes Tavares Neto

**Nome do curso de graduação:**

Engenharia de produção

**Departamento/Centro:**

Departamento de Engenharia de Produção/CCET

**Período de vigência da bolsa:**

01 de setembro de 2021 a 31 de agosto de 2022

**São Carlos/2022**

**Resumo:** Por se tratarem de problemas complexos (*NP-Hard*) de sequenciamento da produção, problemas de FSS com muitas ordens de serviço geralmente têm suas soluções obtidas por meio de algoritmos heurísticos ou meta-heurísticos, como é o caso do algoritmo genético, por serem capazes de fornecer boas soluções em tempo aceitável. Esse estudo investiga os principais operadores de *crossover* utilizados na resolução de problemas de *Flow Shop Scheduling* (FSS) por meio da implementação de um algoritmo genético. Os objetivos foram abordados em três fases: determinação dos principais operadores de *crossover*, por meio de uma análise da literatura, implementação do algoritmo genético e análise e comparação dos resultados obtidos. A pesquisa apresenta a revisão e a descrição dos operadores de *crossover* mais populares na literatura nos últimos anos, assim como resultados quantitativos obtidos pelo algoritmo genético para comparação de desempenho entre esses operadores. A comparação revelou que quatro operadores de *crossover* apresentaram desempenhos superiores de acordo com o critério de minimização de *makespan*: *order-based*, *position-based*, *two-point* e *sequence-based*.

**Palavras-chave:** *Scheduling*; *flow shop*; algoritmo genético; *crossover*.

## 1. Introdução

Problemas relacionados a *scheduling* tratam de processos de tomada de decisões que lidam com a alocação de recursos para conclusão de tarefas em um tempo determinado, com objetivo de otimizar uma ou mais métricas (Pinedo, 2008). Considerando a finitude dos recursos empregados nas atividades humanas, observa-se a necessidade de eficiência na alocação destes para a otimização da operação em diversas áreas, desde o *scheduling* aplicado a procedimentos de pousos e decolagens em aeroportos, até o *scheduling* de cirurgias, com objetivo de minimizar o tempo médio de recuperação dos pacientes (Wang *et al.*, 2020). Nesse contexto, há muitos outros problemas complexos largamente abordados na literatura científica, que oferecem milhões de possíveis soluções, e que necessitam de métodos mais elaborados para sua resolução.

Um desses problemas complexos é o *scheduling* em *flow shop (FSS – Flow Shop Scheduling)*, *layout* de produção que, de acordo com Pinedo (2008), se caracteriza por *m* máquinas, dispostas em série, nas quais são processadas cada uma das *n* tarefas envolvidas em uma sequência de produção pré-determinada pela programação de produção. Todas as tarefas possuem tempos de processamento distintos em cada uma das máquinas, sendo necessária, para resolução do problema, a determinação de uma ou mais sequências de produção que otimizem os objetivos estipulados. A representação desse *layout* se popularizou na linha de montagem,

no início do século XX, na qual os materiais eram levados por uma esteira para operários ou máquinas, dispostos em uma sequência pré-determinada, que realizariam as tarefas necessárias à confecção dos produtos.

Uma função objetivo muito utilizada na resolução de problemas de FSS é a de minimização do *makespan*, definido como o tempo total de processamento, isto é, o período transcorrido entre o início do primeiro trabalho da primeira máquina até a conclusão da última tarefa na máquina *m*.

Por apresentarem complexidade *NP-Hard* (Garey *et al.*, 1976), não existe um algoritmo que obtenha soluções ótimas para tais problemas em tempo polinomial, e, por isso, a abordagem de resolução para o *scheduling* em *flow shop* é realizada por meio da utilização de algoritmos heurísticos e meta-heurísticos, que possibilitam a obtenção de boas soluções em tempo aceitável, apesar de não garantirem soluções ótimas. Dentre as meta-heurísticas mais populares na literatura está o algoritmo genético, pertencente à classe dos algoritmos evolutivos, que utilizam estratégias inspiradas nos mecanismos de evolução biológica para explorar o espaço de soluções. Uma dessas estratégias é realizada pelo operador de *crossover*, responsável pela combinação da informação de duas soluções para a formação de novas soluções. Em uma revisão preliminar da literatura notamos que há diversos operadores de *crossover*, inspirando o objetivo deste trabalho, que é comparar os resultados obtidos pela implementação desses diferentes operadores para determinar quais possuem os melhores desempenhos.

Para apresentação dos resultados dessa pesquisa, esse documento é estruturado da seguinte maneira: na seção 2 é apresentado o referencial teórico utilizado, na seção 3 temos a descrição do problema, a seção 4 aborda a implementação e análises dos resultados obtidos, as considerações finais se encontram na seção 5.

## 2. Referencial teórico

### 2.1 Scheduling

A quantidade de problemas de *scheduling* é praticamente ilimitada, dado que os recursos, tarefas e objetivos neles envolvidos podem assumir diferentes formas. Podemos pensar na alocação de recursos tão diversos quanto máquinas, equipamentos médicos e processadores, aplicados a tarefas envolvidas na produção de bens de consumo, tratamentos de saúde e execução de programas em um computador, respectivamente. Os objetivos a serem otimizados pelo *scheduling* também podem assumir formas como: minimização do tempo total de processamento, minimização de impactos ambientais, minimização do tempo de setup, minimização do atraso de entregas, entre muitas outras possiblidades (Pinedo, 2008).

Problemas de sequenciamento da produção podem ser descritos, de acordo com Graham *et al.* (1997), por uma notação de três parâmetros, $\alpha|\beta|\gamma$, em que $\alpha$ especifica o ambiente de máquinas, $\beta$ indica as características das tarefas e $\gamma$ diz respeito ao critério de otimização escolhido. Nessa categoria de problemas, temos $n$ tarefas $J_j\ (j = 1, \dots, n)$ que devem ser processadas por $m$ máquinas $M_i (i = 1, \dots, m)$. Nesses termos, podemos descrever o problema abordado na pesquisa.

O ambiente de máquinas, $\alpha = \alpha_1\alpha_2$, tem $\alpha_1 = F$, tratando-se, dessa forma, de um problema de *scheduling* em *flow shop*, no qual cada tarefa $J_j$ segue uma ordem de processamento $O_{ij}$, de acordo com a qual cada uma delas deve ser processada na máquina $M_i$ em um período $p_{ij}$. Nesse caso temos que $\alpha_2 = m$, sendo $\alpha = F_m$.

Quanto às características das tarefas temos $\beta = prmu$, isto é, a sequência de processamento das tarefas é a mesma para todas as máquinas disponíveis. As soluções aceitas são, então, restritas à permutação da quantidade de tarefas ($n$), havendo $n!$ possíveis soluções em problemas de *flow shop* permutacional, como os abordados nessa pesquisa. Nesse caso, a ordem de processamento pode ser descrita apenas por $O_j$, pois sabemos que a sequência será a mesma para todas as máquinas.

O critério de otimização envolve a minimização do *makespan* de uma sequência de tarefas, em que $C_{ij}$ é o tempo de conclusão de $J_j$ em $M_i$. Portanto, $\gamma = C_{max} = max(C_{ij})$.

Temos assim que o problema abordado é descrito pela notação $F_m|prmu|C_{max}$, tratando-se do caso permutacional do problema de *scheduling* em *flow shop* com objetivo de minimizar o *makespan*.

## 2.2 Algoritmo genético

O algoritmo genético é uma meta-heurística que, conforme proposta de Holland (1992), se inspira no processo de seleção natural para obter soluções de problemas de otimização por meio de operadores de seleção, *crossover* e mutação. Para os propósitos desse projeto foi implementado um algoritmo genético no qual não ocorre mutação, com o intuito de limitar a influência de outros operadores na análise comparativa dos operadores de *crossover*.

### 2.2.1 Codificação da solução

As soluções foram representadas como vetores *n*-dimensionais que devem conter uma permutação das *n* tarefas para serem válidas, respeitando as condições de que todos os trabalhos devem ser processados em todas as máquinas e de que uma máquina não pode realizar o mesmo

trabalho mais de uma vez. Em um algoritmo genético, podemos chamar essa representação vetorial da solução de cromossomo, no qual cada elemento é chamado de gene. O conjunto de soluções disponíveis a cada iteração do algoritmo é chamado de geração.


**Figura 1.** Representação de uma solução (cromossomo) para um problema com 5 tarefas. As tarefas são processadas da esquerda para a direita, até atingir a última posição do cromossomo. Nesse caso, o número total de possíveis ordens de processamento é $5! = 120$.

### 2.2.2 Seleção

Seguindo os trabalhos de Zou et al. (2021), Wang et al. (2020), Dan et al. (2021), Xiong et al. (2021), Lu & Qiao (2021), Yu et al. (2020), Branda et al. (2020) e Farooq et al. (2021), utilizamos o operador de *roulette-wheel selection*, o qual atribui uma probabilidade de seleção, proporcional a uma medida de adaptabilidade, para cada uma das soluções. Para o critério de minimização do *makespan* foi necessário calcular o valor auxiliar de *fitness*, aqui definido como a diferença absoluta entre o valor do *makespan* de uma solução e o maior valor de *makespan* da geração, garantindo que as melhores soluções teriam os maiores valores de *fitness*, e, portanto, maiores probabilidades de serem selecionadas. A probabilidade de seleção para cada solução foi definida como a razão entre o valor de *fitness* da solução e o somatório dos valores de *fitness* para todas as soluções da geração. Com a finalidade de evitar a perda de boas soluções ao longo das gerações foi implementada uma estratégia elitista, que garantia a presença das duas melhores soluções de cada geração na geração seguinte. Nesse contexto, as soluções selecionadas para realizar o *crossover* são chamadas soluções pais.


**Figura 2.** Fluxograma do operador de *roulette-whell selection.* O operador recebe uma população, calcula o *fitness* de cada uma das soluções, define a probabilidade de seleção para cada uma delas e seleciona aleatoriamente duas soluções pais com base nas probabilidades estabelecidas.

### 2.2.3 Crossover

O operador de *crossover* é aquele que recebe as duas soluções pais e, após combinar suas informações genéticas, retorna duas soluções filhas. Para determinar os operadores de *crossover* a serem implementados, foram pesquisados artigos na base da CAPES considerando trabalhos publicados nos últimos dois anos. A partir desses artigos, selecionamos os operadores mais utilizados. Foram selecionados os 11 seguintes operadores: *order-based, position-based, partially-mapped (PMX), one-point, two-point, two-point permutation (two-point crossover 2), order crossover 2 (OX2),* linear, *sequence-based, loop-based* e *two-cut PTL*. Cada um desses difere quanto ao critério de seleção e transmissão da informação genética dos pais para os filhos. A seguir veremos a descrição de cada um deles.

(1) *Order-based crossover*

Nesse operador há a transferência direta de uma sequência ordenada de genes, definida aleatoriamente, a partir de um dos pais, para uma solução filha. Os genes faltantes na solução filha são então preenchidos conforme aparecem no cromossomo do outro pai (Saraçoğlu *et al.*, 2021).

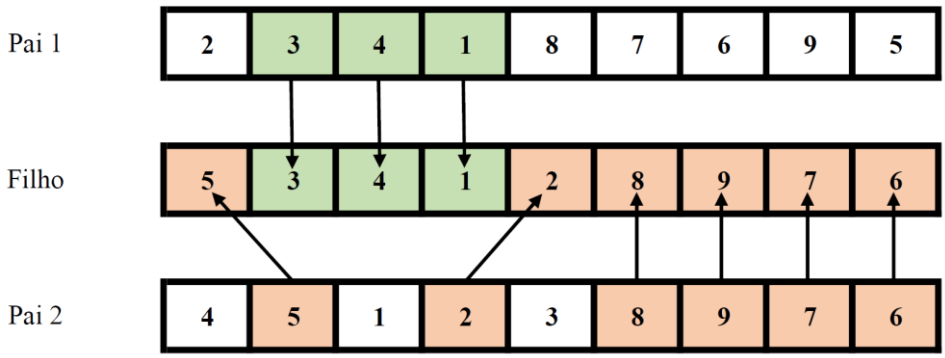


**Figura 3.** Exemplo do *order-based crossover*.

(2) *Position-based crossover*

A transferência de informações para o filho ocorre diretamente a partir de um conjunto aleatório de posições selecionado em um dos pais. Os genes faltantes na solução filha são preenchidos conforme aparecem no outro pai (Saraçoğlu *et al.*, 2021).

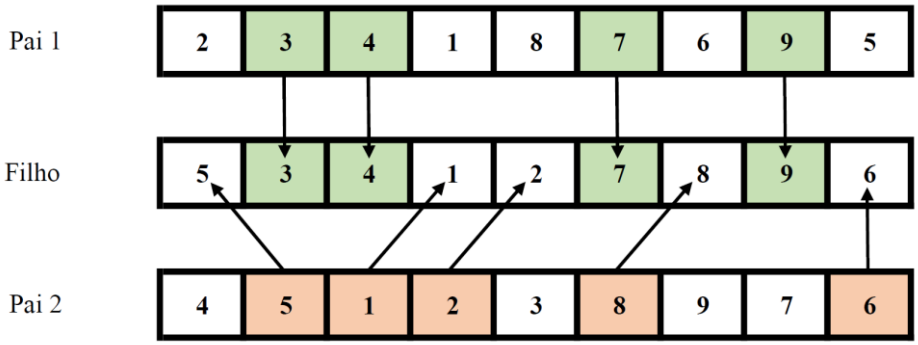


**Figura 4.** Exemplo do *position-based crossover*.

(3) *Partially-mapped crossover (PMX)*

Trata-se de um operador utilizado para gerar novas soluções em problemas que não permitem a repetição de genes, como é o caso do *flow shop* permutacional. Inicialmente são definidos dois pontos de corte, válidos para ambas as soluções pais. Realiza-se o mapeamento dos genes entre os pontos de corte de ambos os pais, trocando as regiões entre os pontos para gerar as soluções filhas. Os genes repetidos que estão fora da região de corte das soluções filhas são então substituídos de acordo com o mapeamento realizado anteriormente, garantindo que todas as soluções obtidas serão permutações válidas (Yuan *et al.*, 2020).

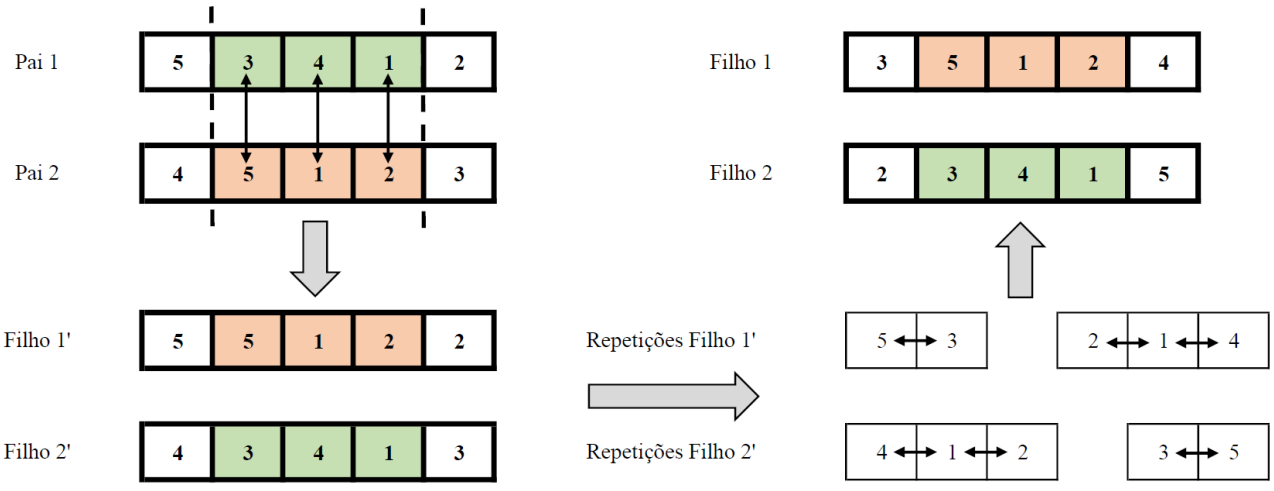


**Figura 5.** Exemplo do *partially-mapped crossover (PMX).*

(4) *One-point crossover*

É selecionado aleatoriamente um ponto de corte válido para ambas as soluções pais. A informação do início de um cromossomo pai até o ponto de corte é transferida para uma solução

filha. O restante dos genes são preenchidos conforme aparecem no cromossomo do outro pai (Farooq *et al.*, 2021).

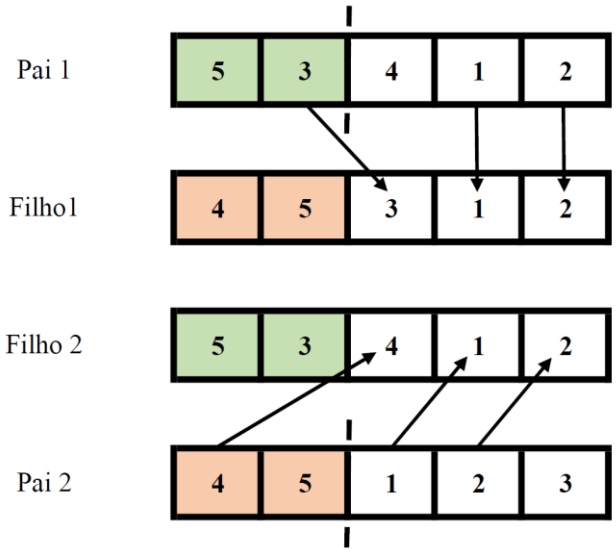


**Figura 6.** Exemplo do *one-point crossover*.

(5) *Two-point crossover*

Ocorre a seleção aleatória de dois pontos de corte, válidos para ambas as soluções pais. A informação entre os dois pontos na solução de um dos pais é transferida para o cromossomo de um filho. Os genes faltantes são preenchidos conforme aparecem no outro pai (Wang *et al.*, 2020). A principal diferença em relação ao *order-based crossover* implementado é que no *two-point crossover* são transferidas as mesmas posições de ambos os pais, ao passo que no *order-based* cada um dos pais transfere uma sequência genética definida de maneira independente.

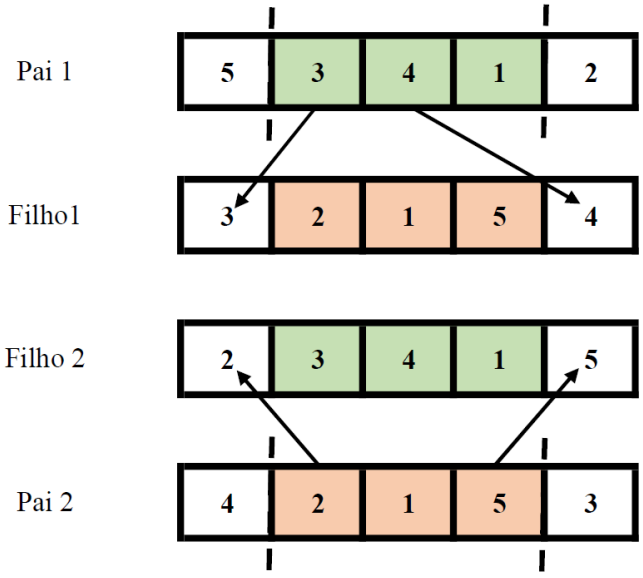

**Figura 7.** Exemplo do *two-point crossover*.

(6) *Two-point permutation crossover (two-point crossover 2)*

Nessa variação do *two-point crossover* os genes transferidos diretamente pelo cromossomo pai para um dos filhos se encontram nas extremidades da região entre os dois pontos de corte, e os genes faltantes, no centro da solução filha, são preenchidos conforme aparecem no outro pai (Boufellouh & Belkaid, 2020).

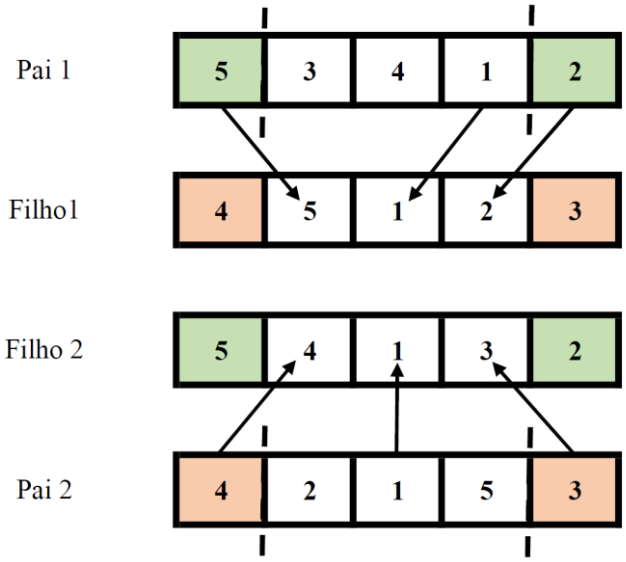


**Figura 8.** Exemplo do *two-point permutation crossover.*

(7) *Order crossover 2 (OX2)*

Como variação do *order-based crossover* temos duas sequências ordenadas de genes que serão transferidas diretamente de uma solução pai para uma filha. A primeira sequência vai do início do cromossomo até o primeiro ponto de parada, escolhido aleatoriamente, a segunda sequência vai do segundo ponto até o final do cromossomo. Os genes faltantes, que ficarão nas posições centrais, são então preenchidos conforme aparecem no outro pai (Dan *et al.*, 2021). A diferença entre este operador e o *two-point permutation crossover* implementado é que no *OX2* os pais podem ter pontos de paradas diferentes para seleção da informação utilizada na formação de cada solução filha.

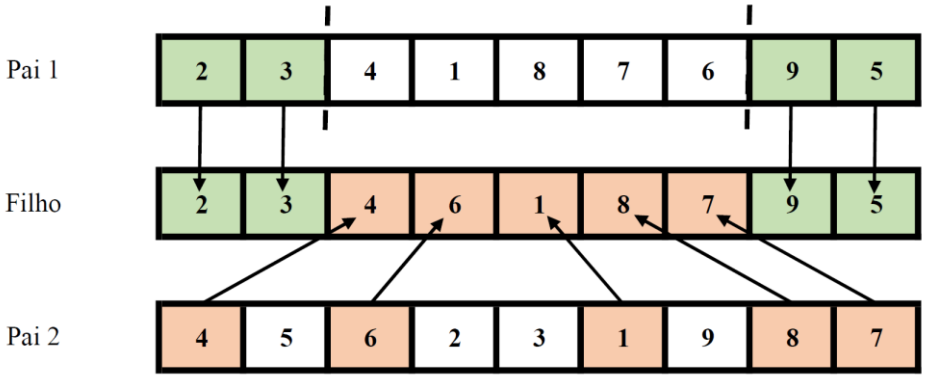


**Figura 9.** Exemplo do *order crossover 2*.

(8) *Linear crossover*

Nesse operador as soluções filhas são inicializadas como cópias de cada um dos pais. São determinados aleatoriamente dois pontos de corte, aplicados a ambos os pais, e os genes entre os pontos de corte de cada pai são removidos daquele filho criado como cópia do outro pai, sendo então reinseridos na mesma ordem em que aparecem na região entre os pontos de corte (Han *et al.*, 2020).

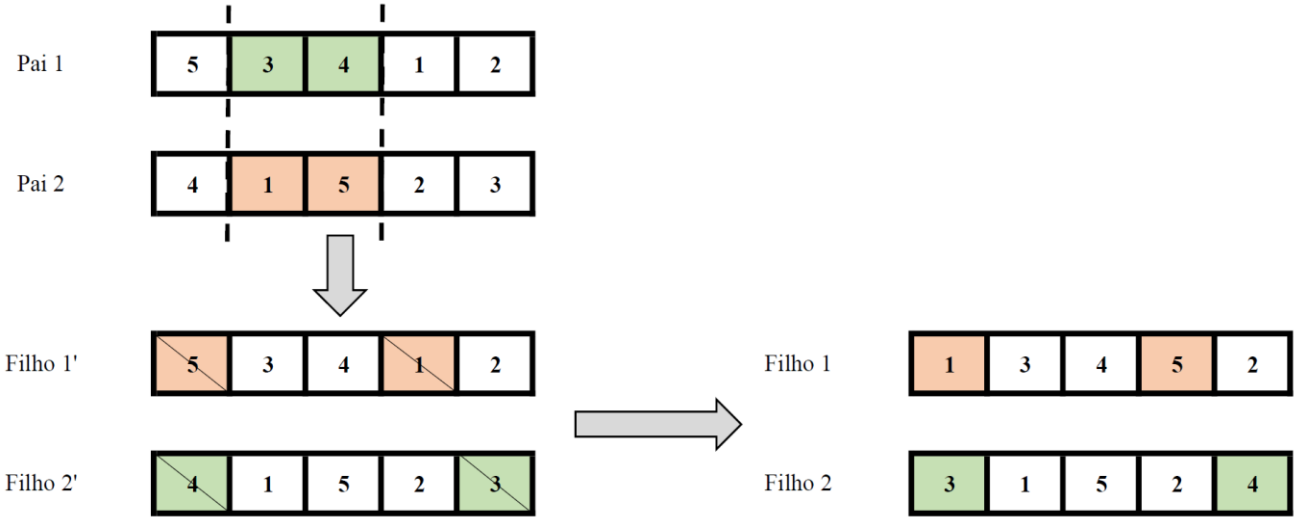


**Figura 10.** Exemplo do *linear crossover*.

(9) *Sequence-based crossover*

Seleciona-se metade dos genes de um dos pais aleatoriamente e ordena suas posições de ocorrência. Esses mesmos genes são localizados no outro pai e suas posições de ocorrências também são sequenciadas. Para geração das soluções filhas é feito o *crossover* desses genes selecionados de acordo com o mapeamento de suas posições sequenciadas (Wu *et al.*, 2020).

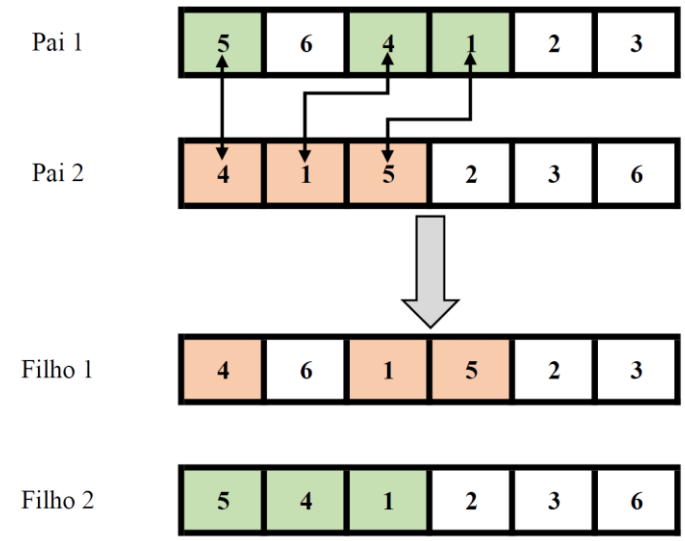


**Figura 11.** Exemplo do *sequence-based crossover*.

(10) *Loop-based crossover*

Inicializam-se uma variável de posição *p* e um vetor de posições vazio *K*. Enquanto o gene na posição *p*, iniciando com *p=1*, do segundo pai for diferente do primeiro gene do primeiro pai, *p* é atualizado com a posição do gene do primeiro pai que corresponde ao gene na posição *p* do segundo. Os valores atualizados de *p* vão sendo armazenados no vetor *K*. Por fim, a formação da solução filha ocorre recebendo os genes do primeiro pai nas posições contidas em *K*, e, no restante das posições, recebendo os elementos do segundo pai (Wu *et al.*, 2020).

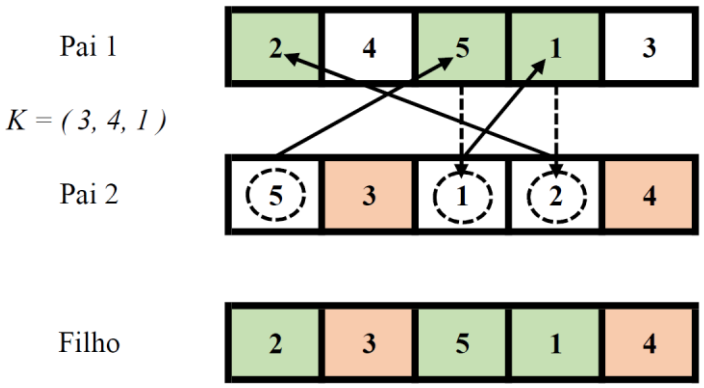


**Figura 12.** Exemplo do *loop-based crossover*. Nesse caso, iniciando com $p = 1$ temos o gene 5 no Pai 2, que é diferente de 2, gene da primeira posição do Pai 1. Assim,$p$ recebe o valor de posição do gene 5 no Pai 1, que é 3, valor armazenado em $K$. Para $p = 3$ temos o gene 1 no Pai 2, correspondente ao elemento de posição 4, armazenada em $K$, no Pai 1. Por fim, $p = 4$

apresenta o gene 2 no Pai 2, que é igual ao primeiro gene do Pai 1, atendendo à condição de parada do *loop*, e armazenando a posição 1 no vetor *K*.

(11) *Two-cut PTL crossover*

Ocorre a determinação aleatória de dois pontos de corte no cromossomo de um dos pais, os genes entre os pontos são movidos para o final de uma das soluções filhas e para o início da outra. Os genes faltantes nas soluções filhas são preenchidos conforme aparecem no outro pai (Yüksel *et al.*, 2020). Esse operador permite que até mesmo duas soluções pais idênticas gerem soluções distintas (Pan *et al.*, 2007).

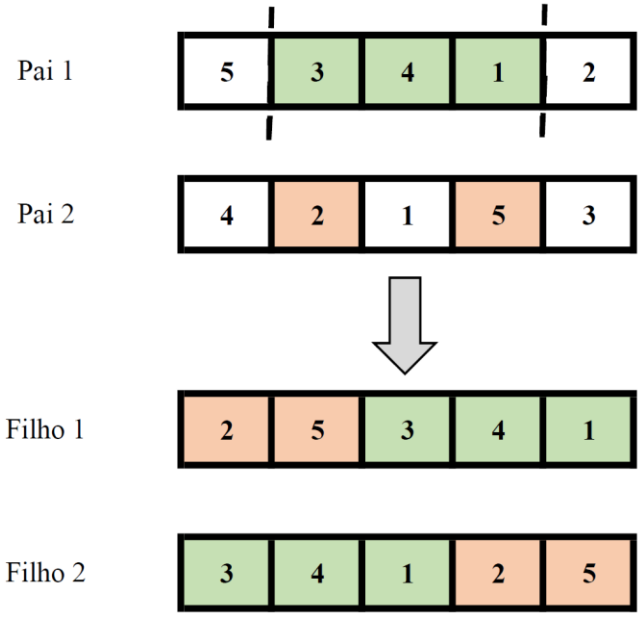


**Figura 13.** Exemplo do *two-cut PTL crossover*.

### 2.2.4 Estrutura do algoritmo

O algoritmo genético desenvolvido inicializa aleatoriamente uma população de soluções para compor a primeira geração. Todas as gerações são ordenadas de acordo com o valor da função objetivo. A partir daí, para cada par de soluções necessárias à composição da geração seguinte, é utilizado o operador de *roulette-wheel selection*, selecionando duas soluções com probabilidade de realizar *crossover*. Em toda geração as duas melhores soluções, aquelas com os menores valores de *makespan*, são transferidas diretamente para a próxima geração. O *crossover* é realizado por meio de algum dos operadores supracitados, sendo este definido previamente, não havendo possibilidade de uso simultâneo de dois ou mais operadores. O

critério de parada é definido de acordo com o número máximo de gerações. Por fim, retorna a melhor solução da última geração.

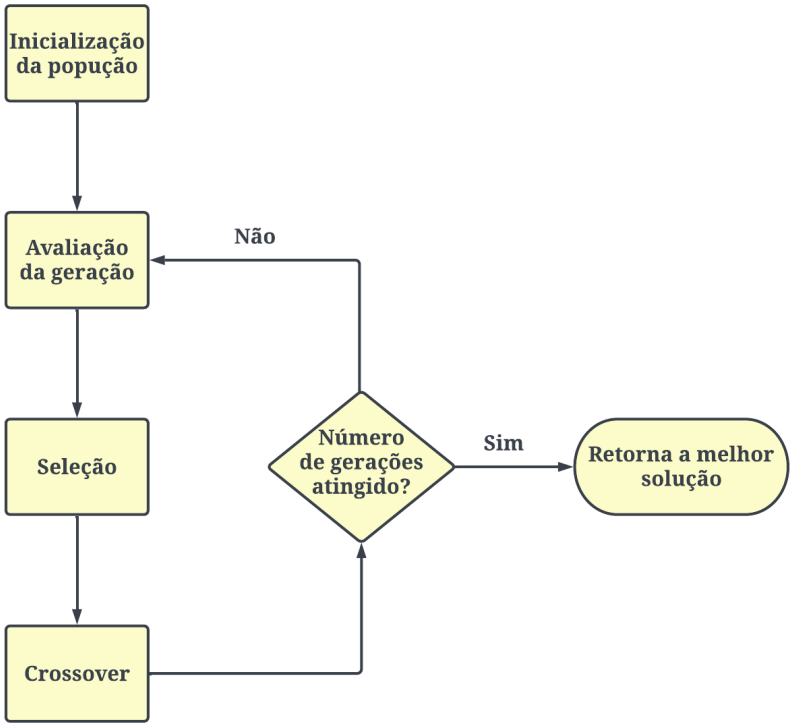


**Figura 14.** Fluxograma do algoritmo genético desenvolvido.

## 3. Descrição do problema

Os problemas de FSS permutacionais buscam encontrar a ordem de processamento $O_j$, em que $j = (1, \dots, n)$ representa o índice das tarefas, das tarefas $J_j$ que minimiza o tempo total de processamento. Sendo $p_{ij}$, em que $i = (1, \dots, m)$ é o índice das máquinas, o tempo de processamento de $J_j$ na máquina $M_i$, o tempo de conclusão de uma tarefa em uma máquina é determinado pela expressão $C_{ij} = max\left(C_{(i-1)j}, C_{iS_{(o_j-1)}}\right) + p_{ij}$, em que $S_k$, $k = (1, \dots, n)$, refere-se à tarefa na posição $k$ dentro da solução avaliada. O *makespan* é, então, $C_{max} = max\left(C_{ij}\right)$.

Dada a natureza permutacional do problema, temos que a ordem de processamento das tarefas é sempre a mesma para todas as máquinas, e o processamento ocorre nas máquinas $1, \dots, m$, necessariamente nesta ordem. Neste caso a preempção não é admitida, isto é, uma tarefa não pode ser interrompida durante a sua execução, possibilitando a execução de outras

tarefas, para ser concluída posteriormente. Há também as restrições de que cada máquina pode trabalhar em apenas uma tarefa por vez, e de que a mesma tarefa não pode ser processada simultaneamente em máquinas diferentes.

A estrutura do problema pode ser visualizada em um diagrama de Gantt:

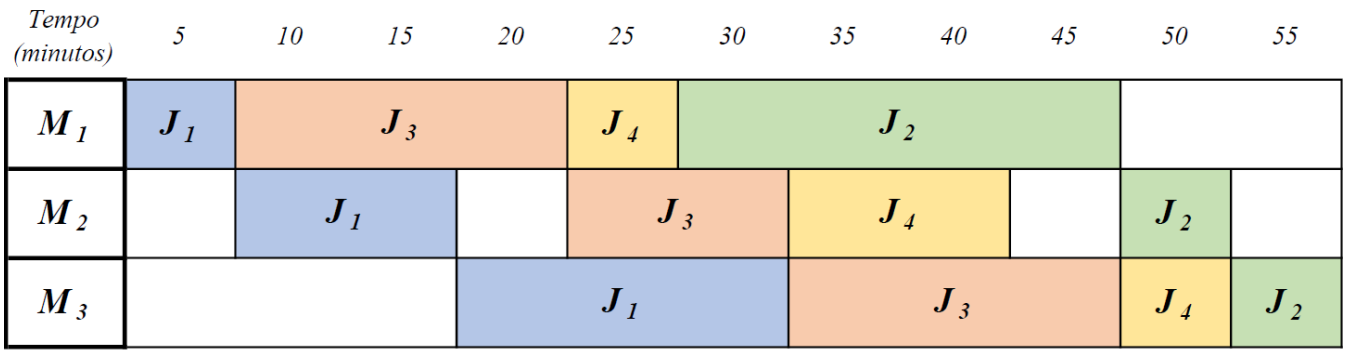


**Figura 15.** O diagrama exemplifica um esquema de *scheduling* permutacional em ambiente de *flow shop*. Cada uma das $J_j$, $j = (1, 2, 3, 4)$, deve ser processada em cada $M_i$, $i = (1, 2, 3)$. A ordem de processamento das tarefas é a mesma em todas as máquinas. O processamento de $J_j$ só pode ser iniciado em $M_i$ quando esta está vazia e $J_j$ foi concluída em $M_{i-1}$. O *makespan* nesse caso é $C_{32} = max(C_{22}, C_{3S_3}) + p_{32} = max(C_{22}, C_{34}) + p_{32}$, isto é, o tempo de conclusão de $J_2$ em $M_3$. Sendo $C_{22} = C_{34} = 50$ e $p_{32} = 5$, o *makespan* dessa sequência de produção totalizou 55 minutos.

## 4. Implementações e Análise de resultados

Avaliamos soluções para instâncias de teste de problemas de *flow shop* permutacional com $n = (20, 50, 100, 200, 500)$ e $m = (5, 10, 20)$ conforme publicadas por Taillard (1993), totalizando 120 instâncias de teste. Estes problemas foram formados por meio de um gerador de números aleatórios que fornece o tempo de processamento $(p_{ij})$. Para obtenção de boas soluções o autor utilizou métodos heurísticos baseados nas técnicas de *taboo search*, que alcançam bons resultados ao custo de grande tempo de processamento. As instâncias de teste publicadas foram escolhidas por grau de dificuldade, oferecendo os problemas mais complexos provenientes do gerador de números aleatórios desenvolvido.

Em cada execução do algoritmo devem ser definidos os parâmetros de tamanho populacional, número máximo de gerações e probabilidade de crossover. O *software irace* (López-Ibáñez *et al.*, 2016) foi utilizado para auxiliar na escolha dos parâmetros do algoritmo genético.

Aplicamos todos os operadores de *crossover* desenvolvidos à resolução de cada uma das instâncias de teste. Os valores de *makespan* obtidos ao utilizar cada um desses operadores possibilitaram a análise do erro relativo a partir dos menores valores de *makespan* documentados para as instâncias de teste. A comparação entre os erros relativos de cada um dos operadores, assim como o erro relativo ao utilizarmos soluções aleatórias, caso não fossem implementadas estratégias de *crossover*, é apresentada a seguir:

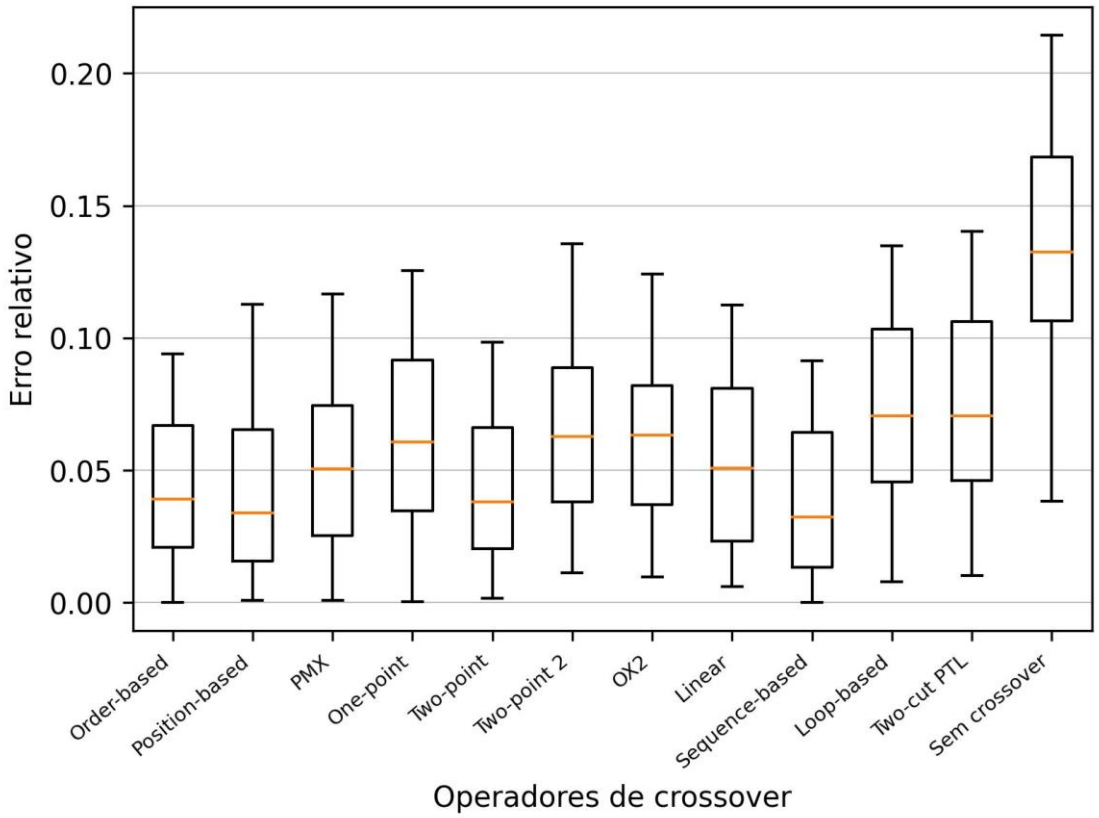


**Figura 16.** *Boxplot* dos erros relativos de cada um dos operadores de *crossover* implementados aplicados a todas as instâncias de teste. Os parâmetros utilizados foram: tamanho populacional de 194 soluções, 133 gerações e taxa de *crossover* de 72,08%.

É possível analisar visualmente, de acordo com a Figura 14, que quatro dos operadores desenvolvidos se destacam por apresentarem a mediana do erro relativo abaixo de 5%: *order-based*, *position-based*, *two-point* e *sequence-based.* A superioridade do desempenho das operações de *crossover* realizadas por meio do algoritmo genético em relação ao uso de soluções aleatórias é confirmada pela disparidade entre o erro relativo dos operadores, todos com mediana abaixo de 10%, e o erro relativo no caso da implementação sem *crossover*, que ultrapassa esse patamar. Restringindo a análise aos operadores com melhor desempenho de

acordo com o objetivo estabelecido de minimização do *makespan*, podemos visualizar a comparação entre eles:

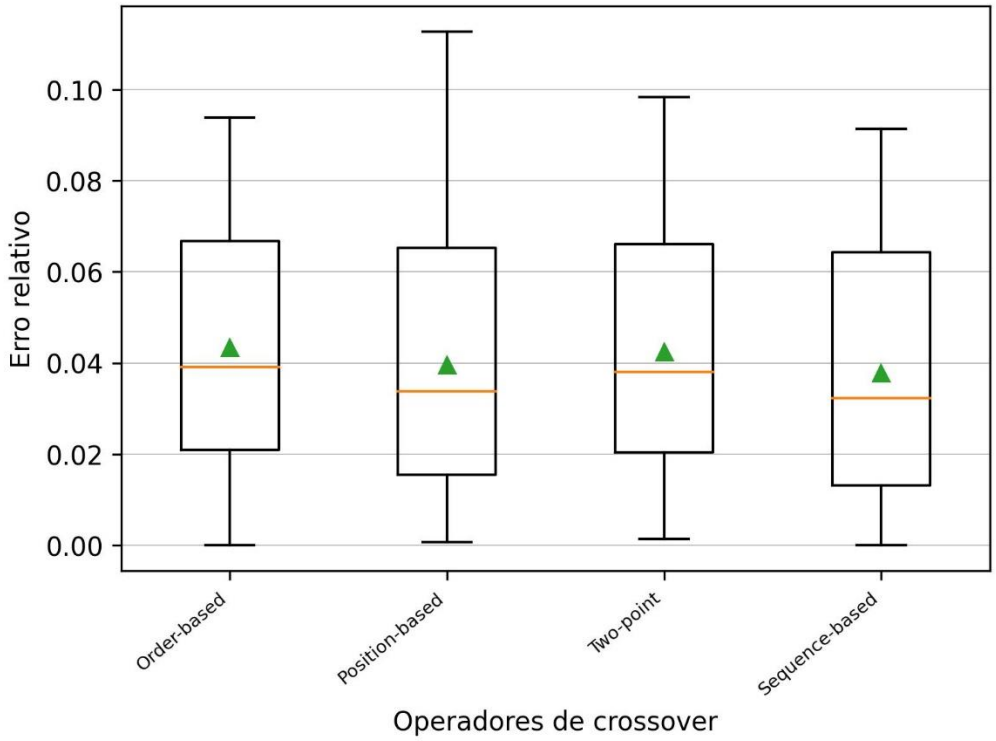


**Figura 17.** *Boxplot* dos erros relativos dos melhores operadores de *crossover*. Foram selecionados aqueles que apresentaram a mediana do erro relativo inferior a 5%.

Entre os melhores operadores vemos que o *sequence-based crossover* apresenta as menores mediana, média e valor máximo do erro relativo. Além disso, apenas o *order-based* e o *sequence-based crossover* foram capazes de obter soluções iguais aos melhores valores documentados para as instâncias de teste, apresentando valor mínimo do erro relativo nulo. O desempenho desse operador, no entanto, não é significativamente superior à dos outros três avaliados, sendo impossível determinar a superioridade de algum deles.

**Tabela 1.** Descrição estatística do erro relativo dos melhores operadores de *crossover*.

| **Operadores** | **Order-based** | **Position-based** | **Two-point** | **Sequence-based** |
|---|---|---|---|---|
| **Número de testes** | 120 | 120 | 120 | 120 |
| **Média do erro relativo** | 0,0433 | 0,0395 | 0,0423 | 0,0378 |
| **Desvio padrão** | 0,0258 | 0,0268 | 0,0252 | 0,0276 |

| | | | | |
|---|---|---|---|---|
| **Min** | 0 | 0,0007 | 0,0014 | 0 |
| **Mediana** | 0,0391 | 0,0338 | 0,0381 | 0,0323 |
| **Max** | 0,0938 | 0,1126 | 0,0984 | 0,0913 |

É também possível avaliar o desempenho de cada operador quanto à complexidade computacional, comparando o tempo necessário para a resolução de instâncias de teste ao utilizarmos cada um deles.

**Tabela 2.** Análise descritiva do tempo de resolução, em segundos, dos operadores de *crossover*. Foram utilizadas 12 instâncias de teste, uma para cada dimensão de problema disponível, utilizando como parâmetros 194 soluções, 133 gerações e taxa de *crossover* de 72,08%.

| **Operadores** | **Order-based** | **Position-based** | **PMX** | **One-point** | **Two-point** | **Two-point 2** |
|---|---|---|---|---|---|---|
| **Testes** | 12 | 12 | 12 | 12 | 12 | 12 |
| **Tempo médio** | 165,5 | 220,6 | 164,0 | 178,3 | 184,1 | 184,4 |
| **Desvio padrão** | 190,8 | 310,1 | 185,0 | 229,4 | 243,6 | 246,1 |
| **Min** | 50,5 | 54,1 | 51,5 | 49,7 | 49,7 | 51,8 |
| **Mediana** | 99,8 | 118,2 | 99,7 | 105,0 | 107,9 | 100,5 |
| **Max** | 730,2 | 1161,2 | 708,2 | 872,9 | 925,1 | 934,9 |

| **Operadores** | **OX2** | **Linear** | **Sequence-based** | **Loop-based** | **Two-cut PTL** | **Sem crossover** |
|---|---|---|---|---|---|---|
| **Testes** | 12 | 12 | 12 | 12 | 12 | 12 |
| **Tempo médio** | 187,8 | 182,7 | 180,7 | 158,9 | 164,1 | 148,5 |
| **Desvio padrão** | 262,5 | 248,3 | 246,4 | 180,1 | 183,2 | 168,6 |
| **Min** | 48,9 | 47,9 | 49,1 | 48,1 | 50,5 | 46,2 |
| **Mediana** | 100,0 | 97,6 | 94,8 | 97,3 | 98,9 | 84,8 |
| **Max** | 992,6 | 940,8 | 932,6 | 691,4 | 703,2 | 644,0 |

Os dados da Tabela 2 mostram que os menores tempos de resolução das instâncias de teste são atribuídos à implementação em que não ocorre o *crossover*, devido à sua menor complexidade computacional. O *position-based crossover* se destaca pelos maiores valores de

média, mediana, desvio-padrão e tempos mínimo e máximo, apresentando-se como o operador mais complexo entre os desenvolvidos.

Portanto, dentre os 11 operadores de *crossover* avaliados, quatro deles apresentaram desempenhos superiores de acordo com o critério de minimização de *makespan*: *order-based, position-based, two-point* e *sequence-based*, todos com mediana e média do erro relativo inferiores a 5%. Entre esses quatro, o *sequence-based crossover* apresentou as melhores métricas de desempenho conforme a análise do erro relativo, entretanto, a diferença em relação aos outros operadores não foi significativa para determinar o melhor operador entre eles. A análise dos tempos de resolução de cada operador revelou que o *position-based crossover* demanda mais tempo computacional, e, entre os melhores operadores, a resolução por meio do *order-based crossover* foi a mais rápida.

## 5. Considerações finais

O presente relatório apresentou um estudo comparativo de um conjunto de 11 operadores de *crossover* em otimizadores baseados em algoritmos genéticos aplicados a problemas de scheduling em ambientes de *flowshop* permutacional com o objetivo de minimizar o makespan. No inicio desse estudo, uma análise preliminar da literatura nos permitiu elencar 11 operadores de *crossover*: *order-based, position-based, partially-mapped, one-point, two-point, two-point permutation, order crossover 2*, linear, *sequence-based*, *loop-based* e *two-cut PTL*.

Os operadores de *crossover* elencados foram aplicados a todas as instâncias de teste disponíveis em Taillard (1993). A análise estatística do erro relativo permitiu distinguir quatro operadores com desempenho superior aos demais, e, apesar da ligeira vantagem do *sequence-based crossover*, não foi possível determinar a superioridade entre estes operadores. Apesar do bom desempenho, o *position-based crossover* demandou o maior tempo computacional entre todos os operadores, sendo preferível a implementação de operadores com desempenho similar e menor complexidade. Entre os quatro melhores operadores, o *order-based crossover* foi o que forneceu bons resultados em menor tempo.

Entende-se como principal limitação da presente pesquisa o fato de que só aplicamos os diferentes operadores em problemas de *flowshop* permutacional e com um único objetivo, o de minimização do *makespan*. Assim sendo, um dos próximos passos dessa pesquisa seria aplicar os algoritmos desenvolvidos em outros ambientes produtivos e com outros objetivos.

Além disso, futuros trabalhos na área podem expandir a análise a outros operadores de *crossover*, incluir estratégias de hibridação de operadores e o estudo de interações com operadores de mutação.

**Referências**


ABDEL-BASSET, M. et al. A Simple and Effective Approach for Tackling the Permutation Flow Shop Scheduling Problem. *Mathematics*, v. 9, n. 3: 270, 2021. https://doi.org/10.3390/math9030270

BOUFELLOUH, R.; BELKAID, F. Bi-objective optimization algorithms for joint production and maintenance scheduling under a global resource constraint: Application to the permutation flow shop problem. *Computers and Operations Research*, v. 122: 104943, 2020. https://doi.org/10.1016/j.cor.2020.104943

BRANDA, A. et al. Metaheuristics for the flow shop scheduling problem with maintenance activities integrated. *Computers & Industrial Engineering*, v. 151: 106989, 2021. https://doi.org/10.1016/j.cie.2020.106989

DAN, Y.; LIU, G.; FU, Y. Optimized flowshop scheduling for precast production considering process connection and blocking. *Automation in Construction*, v. 125: 103575, 2021. https://doi.org/10.1016/j.autcon.2021.103575

DEEP, K; MEBRAHTU, H. New Variations of Order Crossover for Travelling Salesman Problem. *International Journal of Combinatorial Optimization Problems and Informatics*, v. 2, n. 1, p. 2-13, 2011. https://www.redalyc.org/articulo.oa?id=265219618002

FAROOQ, B. et al. Flow-shop path planning for multi-automated guided vehicles in inteligente textile spinning cyber-physical production systems dynamic environment. *Journal of Manufacturing Systems*, v. 59, p. 98-116, 2021. https://doi.org/10.1016/j.jmsy.2021.01.009

GRAHAM, R. L. et al. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. In: HAMMER, P. L.; JOHNSON, E. L.; KORTE, B. H. (Ed.). *Annals of Discrete Mathematics: Discrete Optimization II.* Amsterdã: North-Holland Publishing Company, 1979. p. 287-326.

HAN, W. et al. Multi-objective evolutionary algorithms with heuristic decoding for hybrid flow shop scheduling problem with worker constraint. *Expert Systems with Applications*, v. 168: 114282, 2021. https://doi.org/10.1016/j.eswa.2020.114282

HOLLAND, J.A. **Adaptation in Natural and Artificial Systems**: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. 1. ed. Cambridge (EUA): MIT Press/Bradford Books, 1992.

KURDI, M. A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. *Applied Soft Computing Journal*, v. 94: 106458, 2020. https://doi.org/10.1016/j.asoc.2020.106458

LIN, C. C.; LIU, W. Y; CHEN, Y. H. Considering stockers in reentrant hybrid flow shop scheduling with limited buffer capacity. *Computers & Industrial Engineering*, v. 139: 106154, 2020. https://doi.org/10.1016/j.cie.2019.106154

LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43-58, 2016. https://doi.org/10.1016/j.orp.2016.09.002

LU, H.; QIAO, F. An efficient adaptive genetic algorithm for energy saving in the hybrid flow shop scheduling with batch production at last stage. *Expert Systems*, v. 39, n. 2: 12678, 2021. https://doi.org/10.1111/exsy.12678

PAN, Q. K.; TASGETIREN, M. F.; LIANG, Y. C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, v. 35, n. 9, p. 2807-2839, 2008. https://doi.org/10.1016/j.cor.2006.12.030

PINEDO, M. L. **Scheduling**: Theory, Algorithms, and Systems. 3. ed. Nova Iorque: Springer, 2008.

PRIYA, A.; SAHANA, S. K. Multiprocessor Scheduling Based on Evolutionary Technique for Solving Permutation Flow Shop Problem. *IEEE Access*, v. 8, p. 53151-53161, 2020. https://doi.org/10.1109/ACCESS.2020.2973575

SARAÇOĞLU, İ.; SÜER, G. A.; GANNON, P. Minimizing makespan and flowtime in a parallel multi-stage cellular manufacturing company. *Robotics and Computer-Integrated Manufacturing*, v. 72: 102182, 2021. https://doi.org/10.1016/j.rcim.2021.102182

TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, v. 64, n. 2, p. 278-285, 1993. https://doi.org/10.1016/0377-2217(93)90182-M

TANGOUR, F.; NOUIRI, M.; ABBOU, R. Multi-Objective Production Scheduling of Perishable Products in Agri-Food Industry. *Applied Sciences*, v. 11, n. 15: 6962, 2021. https://doi.org/10.3390/app11156962

WANG, K. et al. Surgery scheduling in outpatient procedure centre with re-entrant patient flow and fuzzy service times. *Omega*, v. 102: 102350, 2021. https://doi.org/10.1016/j.omega.2020.102350

WU, P. et al. An Improved Genetic-Shuffled Frog-Leaping Algorithm for Permutation Flowshop Scheduling. *Complexity*, v. 2020: 3450180, 2020. https://doi.org/10.1155/2020/3450180

XIONG, F. et al. Just-in-time scheduling for a distributed concrete precast flow shop system. *Computers & Operations Research*, v. 129: 105204, 2021. https://doi.org/10.1016/j.cor.2020.105204

YIN, P. Y. et al. Minimizing the Makespan in Flowshop Scheduling for Sustainable Rubber Circular Manufacturing. *Sustainability*, v. 13, n. 5: 2576, 2021. https://doi.org/10.3390/su13052576

YU, C.; ANDREOTTI, P.; SEMERARO, Q. Multi-objective scheduling in hybrid flow shop: Evolutionary algorithms using multi-decoding framework. *Computers & Industrial Engineering*, v. 147: 106570, 2020. https://doi.org/10.1016/j.cie.2020.106570

YUAN, S.; LI, T.; WANG, B. A co-evolutionary genetic algorithm for the two-machine flow shop group scheduling problem with job-related blocking and transportation times. *Expert Systems With Applications*, v. 152: 113360, 2020. https://doi.org/10.1016/j.eswa.2020.113360

YÜKSEL, D. et al. An energy-efficient bi-objective no-wait permutation flowshop scheduling problem to minimize total tardiness and total energy consumption. *Computers & Industrial Engineering*, v. 145: 106431, 2020. https://doi.org/10.1016/j.cie.2020.106431

ZOU, P.; RAJORA, M.; LIANG, S.Y. Multimodal Optimization of Permutation Flow-Shop Scheduling Problems Using a Clustering-Genetic- Algorithm-Based Approach. *Applied Sciences*, v. 11, n. 8: 3388, 2021. https://doi.org/10.3390/app11083388

**Produção técnico-científica**

Apresentação do desenvolvimento do projeto no 28º Congresso de Iniciação Científica (CIC) da Universidade Federal de São Carlos, realizada no dia 25 de março de 2022.

**Autoavaliação**

Considero que minha participação no Programa Institucional de Bolsas de Iniciação Científica do CNPq foi bastante satisfatória. O desenvolvimento do projeto me permitiu o aprofundamento nos conteúdos abordados pela pesquisa, aprimoramento das habilidades de programação e introdução à literatura e método científicos. Com o auxílio de meu orientador foi possível superar os desafios do projeto e cumprir o cronograma proposto para este trabalho.

Lucas Gabriel Malheiros Machado Silva
Bolsista do CNPq

**Avaliação do orientador**