CEEL – ISSN 2595-2221
Universidade Federal de Uberlândia
06 a 10 de Novembro de 2023

UM ESTUDO COMPARATIVO ENTRE O NSGA-II E UM ALGORITMO GENÉTICO PARA A SOLUÇÃO DE UM *JOB-SHOP SCHEDULING PROBLEM*

GABRIEL ANDRADE QUEIROZ*[1] E EDERSON ROSA DA SILVA[1]

[1]FEELT – Universidade Federal de Uberlândia

Resumo **- Os problemas de escalonamento de processos possuem grande valor no estudo de otimização computacional aplicada a diversas áreas como redes de comunicação, infraestrutura, planejamento e logística. Por serem problemas classificados como de grande dificuldade combinatória, a aplicação de métodos heurísticos como algoritmos genéticos é uma boa escolha na busca de soluções. Assim, este estudo tem como objetivo analisar o emprego de algoritmos genéticos, destacando o NSGA-II, na solução do problema de otimização combinatória multiobjetivo JSSP. Os resultados expõem uma comparação de um algoritmo genético genérico ao NSGA-II, demonstrando que ambos são de grande valia na solução de problemas de escalonamento como o JSSP.**

Palavras-Chave **- algoritmo genético, *JSSP*, otimização multiobjetivo, *NSGA-II*.**

A COMPARATIVE STUDY BETWEEN NSGA-II AND A GENETIC ALGORITHM FOR SOLVING A JOB-SHOP SCHEDULING PROBLEM

Abstract **- Scheduling problems have great value in the study of computational optimization applied to several areas such as communication networks, infrastructure, planning and logistics. Because they are problems classified as having huge combinatorial difficulty, the application of heuristic methods such as genetic algorithms is a good choice in the search for solutions. Thus, this study aims to analyze the use of genetic algorithms, highlighting the NSGA-II, in the solution of the multi-objective combinatorial optimization problem JSSP. The results expose a comparison of a generic genetic algorithm to NSGA-II, demonstrating that both are of great value in solving scheduling problems such as JSSP.**

Keywords **- genetic algorithm, JSSP, multi-objective optimization, NSGA-II.**

*gabriel.andrade@ufu.br

I. INTRODUÇÃO

Os problemas de escalonamento de processos em máquinas são encontrados de maneira recorrente em áreas de otimização de processos computacionais como comunicações digitais [1], infraestrutura [2], planejamento de produção [3] e flexibilidade de sistemas fabris [4]. Um dos mais conhecidos problemas de escalonamento é o *Job-Shop Scheduling Problem* (JSSP).

Para compreender o problema estudado, é necessário definir o escalonamento com base nas teorias computacionais. Um procedimento de escalonamento tem por objetivo alocar recursos no tempo necessário, visando executar um conjunto de processos [5]. Ou ainda, o escalonamento é tido como uma atividade de sequenciamento com a preocupação de alocação ótima de recursos limitados a atividades no tempo [6]. Além disso, a alocação de recursos é realizada atendendo a diversas limitações e objetivos [7]. Por fim, destaca-se que cada ação executada requer um conjunto de recursos de capacidade finita, havendo preferência por ordens de execução [8].

De forma geral, o JSSP pode ser descrito como a seguir: existem conjuntos de *jobs*, ou processos, e de máquinas. Cada *job* é constituído por um conjunto de operações, que devem ser processadas durante um determinado período ininterrupto por uma dada máquina. Ainda, cada máquina é capaz de processar apenas uma operação por vez [9]. As sequências de máquinas são fixas para cada *job*, de forma que a solução do problema é encontrar as sequências de *jobs* para cada máquina visando o tempo mínimo de execução. Esta métrica é chamada *makespan* e mede a qualidade da solução encontrada [10].

O JSSP está entre os problemas mais difíceis de otimização combinatória, sendo classificado como um problema do tipo *nondeterministic polynomial-time complete* (NP-*complete*) [11] [12]. Devido a essa dificuldade inerente, implementações com base em métodos heurísticos são tidas como alternativas interessantes, tendo em vista que propõem boas soluções a partir de regras de prioridade para selecionar operações de escalonamento [9].

O objetivo deste artigo é avaliar o uso de algoritmos genéticos como métodos probabilísticos de resolver o JSSP, destacando a aplicação do *Non-Dominated Sorting Genetic*

Algorithm (NSGA-II). Consequentemente, o restante do artigo é organizado como a seguir: a Seção 2 descreve o modelo matemático do JSSP. Na Seção 3, provê-se os conceitos essenciais a respeito de *Multi-Objective Optimization Problem* (MOOP), *Multi-Objective Combinatorial Optimization Problem* (MOCOP), dominância de Pareto, algoritmos genéticos e NSGA-II. Em sequência, a Seção 4 explora a simulação realizada e os resultados obtidos. Por fim, na Seção 5, têm-se as conclusões a respeito do estudo.

II. JSSP

A solução do JSSP leva em consideração a existência de *N jobs* a serem processados por *M* máquinas e assume o seguinte [13] [14]:

- Uma máquina pode processar apenas um *job* por vez.
- A execução de um *job* por uma máquina é chamada de *operação*.
- Uma operação não pode ser interrompida.
- Um *job* é constituído por no máximo *M* operações.
- A sequência de operação de cada *job*, chamada *sequência da máquina*, e o tempo de processamento para cada operação são conhecidos.
- A sequência de operações em cada máquina, chamada *sequência do job*, é desconhecida. O conjunto de sequências de todos os *jobs* é dito ser uma representação simbólica.
- Uma representação simbólica viável é chamada de escalonamento ou *schedule*, configurando o nome do problema.

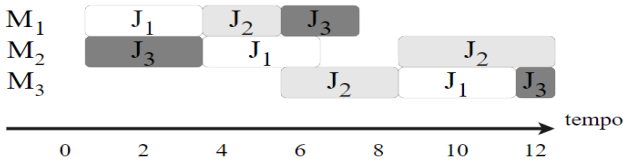
Em [15], Yamada e Nakano definem que o *makespan* mínimo para um JSSP de ordem nxm pode ser descrito como um conjunto de n jobs $\{J_i\}_{1\leq j\leq n}$ a ser processado por um conjunto de m máquinas $\{M_r\}_{1\leq r\leq m}$. Cada *job* apresenta uma sequência de máquinas em que deve ser processado. O processamento do *job* J_j na máquina M_r é chamado *operação* O_{jr}. A operação O_{jr} exige exclusividade na execução pela máquina M_r por um intervalo ininterrupto p_{jr}, seu tempo de processamento. O escalonamento é, de tal modo, o conjunto de tempos finais de execução para cada operação $\{c_{jr}\}_{1\leq j\leq n,1\leq r\leq m}$ que satisfaz a tais imposições. O tempo total para completar todos os *jobs* é chamado *makespan* L. O objetivo de solução ou otimização desse problema é determinar o escalonamento que minimiza L.

Por exemplo, uma possível configuração de um problema 3x3 é demonstrada pela Tabela 1. A partir de tais suposições, uma forma conveniente de representar visualmente a solução é por meio de um diagrama de Grantt como ilustrado na Figura 1.

Tabela 1: Sequência das máquinas e tempos de processamento para os *jobs* de um JSSP 3x3

Operação	O_1	O_2	O_3
J_1	M_1 (3)	M_2 (3)	M_3 (3)
J_2	M_1 (2)	M_3 (3)	M_2 (4)
J_3	M_2 (3)	M_1 (2)	M_3 (1)

Figura 1: Diagrama de Grantt da solução do JSSP 3x3.



III. NSGA-II

Nesta seção, são descritos os conceitos fundamentais acerca do NSGA-II, partindo dos problemas MOOP, MOCOP e dominância de Pareto até os sistemas que constituem o algoritmo: ranqueamento, elitismo, *crowding distance* e seleção.

A. MOOP e MOCOP

De acordo com S. Verma *et al.* [16], um MOOP é constituído por um conjunto de *n* variáveis de decisão, *k* funções objetivo e um conjunto de restrições caracterizadas como *m* desigualdades e *p* igualdades. Assim, tem-se como objetivo de otimização:

$$\text{Min/Max } y = f(x) = (f_1(x), f_2(x), \dots, f_k(x)), k \geq 2 \quad (1)$$

$$\text{Sujeito a } g_i(x) \leq 0, i = 1, 2, \dots, m \quad (2)$$

$$h_j(x) \leq 0, j = 1, 2, \dots, p \quad (3)$$

onde $x = (x_1, x_2, \dots, x_n)$ é um vetor de decisão de *n* dimensões em $X \subseteq R^n$, y é um vetor objetivo de *k* dimensões em R^k, f é a função de mapeamento, enquanto g_i e h_j são, respectivamente, a $i^{ésima}$ e $j^{ésima}$ restrições de desigualdade e igualdade.

Por sua vez, os MOCOPs são casos específicos dos MOOPs, como no caso dos problemas de escalonamento (JSSP), e podem ser formulados da seguinte maneira [16]:

$$\text{Min/Max } y = f(x) = (f_1(x), f_2(x), \dots, f_k(x)), k \geq 2 \quad (4)$$

$$\text{Sujeito a } g_i(x) \leq 0, i = 1, 2, \dots, m \quad (5)$$

$$h_j(x) = 0, j = 1, 2, \dots, p \quad (6)$$

onde $x = (x_1, x_2, \dots, x_n) \in D$ é um vetor de *n* dimensões no espaço de decisão $D = D_1 \, x \, D_2 \, x \dots D_n$ (D_n é o domínio de x_n), enquanto k, m, e p indicam a quantidade de funções objetivo, restrições de desigualdade e igualdade.

B. Dominância de Pareto

Ben-Tal descreve os conceitos matemáticos de solução ótima de Pareto, também chamada de ponto ótimo, solução não dominada ou regra de decisão admissível, em [17]. Assim, introduz-se as seguintes definições:

Dominância de Pareto: *um vetor* $\boldsymbol{a} = (a_1, \dots, a_p)$ *é dito dominante sobre* $\boldsymbol{b} = (b_1, \dots, b_p)$ *se e somente se* $\boldsymbol{a}$ *for parcialmente menor que* $\boldsymbol{b}$*, ou seja,* $\forall i \in \{1, \dots, p\}, a_i \leq b_i \wedge \exists i \in \{1, \dots, p\} : a_i < b_i$.

Veldhuizen e Lamont exemplificam a dominância de Pareto com o seguinte problema de otimização em [18]:

$$\text{minimizar } f(x_u), \; u \in \{a, b, c, d\}, onde$$

$$a = f(x_a) = (3{,}25, 1{,}76, 4{,}67),$$

$$b = f(x_b) = (3{,}25, 1{,}76, 4{,}67),$$

$$c = f(x_c) = (3{,}15, 1{,}76, 4{,}67),$$

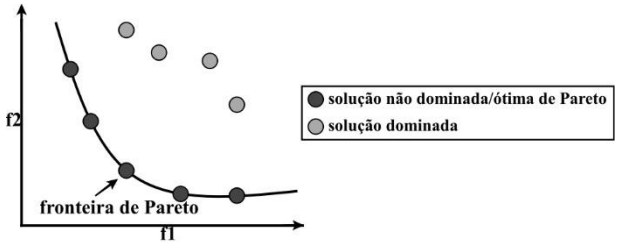
$$d = f(x_d) = (3{,}15, 1{,}76, 4{,}22). \quad (7)$$

Neste caso, a e b são dominados tanto por c e d, enquanto c é dominado por d, e assim, d domina todos os demais vetores.

Otimização de Pareto: *uma solução $x_a \in U$ é dita solução ótima de Pareto se e somente se não existe $x_b \in U$ em que $b = f(x_b) = (b_1, \ldots, b_p)$ domina $a = f(x_a) = (a_1, \ldots, a_p)$.*

Ainda com base no exemplo presente em [18], a solução x_d é a única associada ao conjunto ótimo de Pareto, isto é, x_d é solução ótima de Pareto do conjunto $\{x_a, x_b, x_c, x_d\}$. Os vetores referentes à solução ótima são chamados não-dominados e a seleção de um desses vetores corresponde a uma solução ótima de Pareto. Ainda, ressalta-se que essas soluções podem não apresentar relação aparente além de formarem um conjunto ótimo de Pareto e, quando os vetores são dispostos em um mesmo espaço de escolhas, observa-se uma fronteira de Pareto como ilustrado na Figura 2. Na figura, $f1$ e $f2$ representam as funções que compõem o problema de otimização, ou ainda, as funções objetivo para o algoritmo genético.

Figura 2: Dominância de Pareto.



C. Fundamentos do NSGA-II

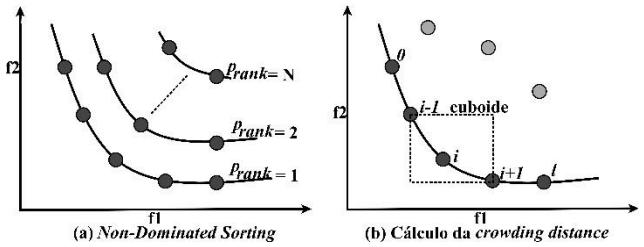
O NSGA-II tem como base quatro conceitos principais, sendo eles: ranqueamento por *Non-Dominated Sorting*, operador de elitismo, *crowding distance* e operador de seleção. As subseções a seguir descrevem de maneira breve tais questões.

i) Non-Dominated Sorting

O procedimento de *Non-Dominated Sorting* classifica os indivíduos da população a partir da dominância de Pareto. Assim, a classificação ou ranqueamento se inicia com a classificação dos membros ditos não-dominados da população inicial como rank 1. Desta forma, esses indivíduos são colocados na primeira fronteira de Pareto e removidos da população inicial. Em sequência, os demais indivíduos são classificados conforme o processo de *Non-Dominated Sorting*, de modo que os indivíduos remanescentes da população são colocados no rank 2 e, consequentemente, na segunda fronteira. Assim, o procedimento continua até que toda a população seja classificada e disposta em fronteiras distintas. A Figura 3(a) ilustra a classificação conforme a lógica de fronteiras.

No procedimento de *Non-Dominated Sorting* descrito por Deb *et al.* [19], para cada população são calculados dois valores: o contador de dominação, n_p, que define a quantidade de soluções que dominam a solução p, e S_p, que designa o conjunto de soluções que a solução p domina.

Figura 3: *Non-Dominated Sorting* e cálculo da *crowding distance*.



Todas as soluções da primeira fronteira terão zero como valor do contador de dominação. Assim, para cada solução p com $n_p = 0$, cada membro q do conjunto S_p tem seu contador de dominação reduzido em um. Com isso, qualquer membro q que tiver n_p reduzido a zero é colocado em uma lista separada Q, sendo classificado como segunda fronteira. Esse processo continua até que todas as fronteiras sejam definidas. O algoritmo de classificação adaptado a partir de [19] é mostrado a seguir:

fast-non-dominated-sort(P)

para cada $p \in P$
  $S_P = \emptyset$
  $n_P = 0$
  para cada $q \in P$
    se $(p \prec q)$ então — se p domina q
      $S_P = S_P \cup \{q\}$ — adiciona q ao conjunto dominado por p
    do contrário $(q \prec p)$ então
      $n_P = n_P + 1$ — incrementa n_p
  se $n_p = 0$ então — p pertence a primeira fronteira
    $p_{rank} = 1$
    $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
$i = 1$ — inicializa o contador de fronteira
enquanto $\mathcal{F}_i \neq \emptyset$
  $Q \neq \emptyset$ — usado para armazenar membros da próxima fronteira
  para cada $p \in \mathcal{F}_i$
    para cada $q \in S_p$
      $n_q = n_q - 1$
      se $n_q = 0$ então — q pertence à próxima fronteira
        $q_{rank} = i + 1$
        $Q = Q \cup \{q\}$
  $i = i + 1$
  $\mathcal{F}_i = Q$

ii) Elitismo

O operador de elitismo mantém as melhores soluções por meio da preservação das mesmas através das gerações. Sendo assim, cada solução não-dominada permanece entre as próximas gerações até que finalmente sejam dominadas.

iii) Crowding distance

Primeiro, é necessário entender a métrica de estimação de densidade. Para obter a estimativa de densidade das soluções acerca de uma solução específica de uma população, deve-se calcular a distância média entre dois pontos em qualquer lado da solução tendo em vista cada um dos objetivos. Assim, a

designada *crowding distance*, $i_{distância}$, serve como uma estimativa do perímetro do cuboide formado pelos vizinhos mais próximos como vértices. A Figura 3(b) ilustra a *crowding distance* da i-ésima solução da fronteira destacada como o tamanho médio do lado do cuboide.

O cálculo da *crowding distance* depende da classificação da população de acordo com cada função objetivo em ordem crescente de magnitude. Assim, para cada função objetivo, as soluções de menor e maior valores de função recebem valor infinito de distância. Todas as demais soluções recebem valor equivalente à diferença normalizada absoluta dos valores das funções de duas soluções adjacentes. Desta forma, o cálculo é realizado para as demais funções objetivo e a *crowding distance* é a soma das distâncias correspondentes a cada objetivo [19].

Deb *et al.* [19], descrevem o algoritmo de cálculo da *crowding distance* da seguinte forma para um conjunto de soluções $\mathcal{I}$:

cálculo-crowding-distance($\mathcal{I}$)	
$l = \|\mathcal{I}\|$	núm. de soluções
para cada i, configurar $\mathcal{I}[i]_{dist.} = 0$	inicia distância
para cada objetivo m	
$\mathcal{I} = classificação(\mathcal{I}, m)$	classificação/rank
$\mathcal{I}[1]_{dist.} = \mathcal{I}[l]_{dist.} = \infty$	considera extremos
para $i = 2$ até $(l-1)$	demais pontos
$\mathcal{I}[i]_{dist.} = \mathcal{I}[i]_{dist.} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{máx.} - f_m^{mín.})$	

$\mathcal{I}[i].m$ refere-se à m-ésima função de objetivo do i-ésimo indivíduo no conjunto $\mathcal{I}$. Por sua vez, os parâmetros $f_m^{máx.}$ e $f_m^{mín.}$ são os valores máximo e mínimo da m-ésima função objetivo. Após todos os membros da população do conjunto $\mathcal{I}$ serem designados a um valor de distância, é possível comparar duas soluções com base em sua proximidade com outras soluções. Assim, uma solução com menor valor desta métrica de distância tem maior lotação por outras soluções [19].

iv) Seleção

Por fim, os autores de [19] definem o operador de comparação de lotação ($\prec_n$) como operador de seleção. Assume-se que cada indivíduo i na população possui dois atributos, sendo eles: o rank (i_{rank}) e a *crowding distance* ($i_{distância}$). Assim, define-se a seguinte relação de ordem $\prec_n$:

$$i \prec_n j \; se \; (i_{rank} < j_{rank})$$
$$ou \; ((i_{rank} = j_{rank}) e (i_{distância} > j_{distância}))$$

Isto é, prefere-se a solução com menor (melhor) rank entre duas soluções de classificações distintas. Do contrário, caso os indivíduos da população pertençam a mesma fronteira, opta-se pela solução que está localizada numa região menos lotada, ou seja, o indivíduo de maior *crowding distance* é selecionado para a próxima geração.

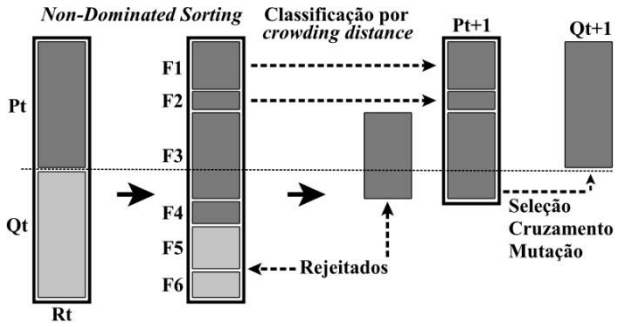
D. Procedimento de operação do NSGA-II

O procedimento do NSGA-II inicia com a criação de uma população inicial P_0. Essa população é classificada com base no processo de *non-dominated sorting*, de forma que cada solução recebe uma classificação. Assim, os operadores gerais de algoritmos genéticos (seleção por torneio, recombinação e mutação) são utilizados para criar uma população de filhos Q_0 de tamanho N. Por sua vez, o elitismo é aplicado comparando a população atual com as melhores soluções não-dominadas encontradas, de forma que as demais iterações são descritas conforme o algoritmo adaptado de [19] indicado a seguir.

$R_t = P_t \cup Q_t$	combina populações de pais e filhos
$\mathcal{F} = fast\text{-}non\text{-}dominated\text{-}sort(R_t)$	$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, todas as fronteiras não-dominadas de R_t
$P_{t+1} = \emptyset \; e \; i = 1$	
até que $\|P_{t+1}\| + \|\mathcal{F}_i\| \leq N$	até completar a população de pais
cálculo-crowding-distance($\mathcal{F}_i$)	calcula a *crowding distance* de $\mathcal{F}_i$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i$	inclui a i-ésima fronteira não-dominada na população de pais
$i = i + 1$	checa a próxima fronteira para inclusão
classifica($\mathcal{F}_i, \prec_n$)	classifica em ordem decrescente conforme $\prec_n$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1:(N - \|P_{t+1}\|)]$	escolhe os primeiros : $(N - \|P_{t+1}\|)$ elementos de $\mathcal{F}_i$
$Q_{t+1} = cria\text{-}nova\text{-}população(P_{t+1})$	usa seleção, cruzamento e mutação para criar a nova população Q_{t+1}
$t = t + 1$	incrementa o contador de gerações

Primeiro, cria-se uma população combinada $R_t = P_t \cup Q_t$, de tamanho $2N$. Em sequência, aplica-se a classificação da população R_t a partir do esquema de *fast-non-dominated-sort*. Assim, as soluções pertencentes ao melhor conjunto, $\mathcal{F}_1$, são as melhores soluções na população R_t e devem ser destacadas. Caso $\mathcal{F}_1$ tenha tamanho inferior a N, todos os membros de $\mathcal{F}_1$ são selecionados para a nova população P_{t+1} e os demais membros são escolhidos com base em sua classificação nas próximas fronteiras. De maneira semelhante, essa etapa é repetida até que não seja possível formar novos conjuntos. Como, em geral, o número de soluções de todos os conjuntos de $\mathcal{F}_1$ a $\mathcal{F}_l$ tende a ser superior ao tamanho da população, as soluções da última fronteira $\mathcal{F}_l$ são classificadas conforme o operador de seleção $\prec_n$ para assegurar que são escolhidos exatamente N membros para a população. Em seguida, a nova população P_{t+1} de tamanho N é utilizada para seleção, cruzamento e mutação para criar uma nova população Q_{t+1} de tamanho N. Esse procedimento continua até completar as condições de parada impostas. A Figura 4 ilustra o procedimento de operação do NSGA-II.

Figura 4: Procedimento do NSGA-II.



IV. SIMULAÇÃO E RESULTADOS

Nesta seção, primeiro são apresentados os dois procedimentos implementados para solução do JSSP: um

algoritmo genético genérico e o NSGA-II. Assim, buscou-se comparar os resultados obtidos de *makespan* ótimo diante das mesmas configurações dos elementos que compõem a estrutura dos algoritmos.

O algoritmo genético é iniciado com uma etapa de codificação, na qual os cromossomos que representam possíveis soluções são criados com base na sequência de operações do JSSP. Cada gene é a codificação de uma operação do conjunto, representando o *job* em questão, a operação e a máquina que realiza o *job*. Em sequência, um conjunto de cromossomos forma a população inicial, que passa por operadores genéticos de cruzamento e mutação para gerar a população de filhos. Assim, é possível avaliar a qualidade do cromossomo com base no cálculo da função de utilidade ou *fitness function*, realizando a etapa de seleção de cromossomo para a formação da nova população. Caso sejam alcançados os critérios de parada, o algoritmo é finalizado adotando a última população como melhor solução.

Em relação ao NSGA-II, inicia-se com a criação da população inicial a partir de um conjunto de cromossomos que seguem a mesma lógica de construção do algoritmo genético. Desta forma, os cromossomos sofrem cruzamento e mutação para gerar a população de filhos. A partir deste ponto, aplica-se a estratégia de elitismo de combinação da população de pais e filhos e são realizadas as etapas de classificação por *Non-Dominated Sorting* e preservação da diversidade por meio do cálculo da *crowding distance*. Por fim, utiliza-se o operador de comparação descrito na Seção III e gera-se uma nova população. Se os critérios de parada forem alcançados, a nova população gerada é tida como melhor solução.

Assim, os dois algoritmos foram implementados seguindo tais procedimentos utilizando a linguagem de programação *Python*. Com base no conjunto de parâmetros descrito na Tabela 2, foram realizadas dez simulações para cada algoritmo, obtendo *makespan* médio de 1201 para o algoritmo genético e de 996 para o NSGA-II. Além disso, os valores de desvio padrão foram, respectivamente, de 23,61 e 21,68 para o algoritmo genético e para o NSGA-II. As Figuras 5 e 6 ilustram o histórico de *makespan* ótimo em função das gerações para os dois algoritmos implementados para os menores valores de *makespan* obtidos. Em ambas as figuras, considerou-se a visualização até a 300ª geração, devido à convergência para o valor ótimo ocorrer antes do total de gerações. Observa-se um ganho médio de 17,07% entre NSGA-II e algoritmo genético para o valor de *makespan* ótimo médio.

A Figura 7 ilustra o diagrama de Grantt para a melhor solução por meio da aplicação do NSGA-II.

V. CONCLUSÕES

Este estudo propõe a avaliação do uso de algoritmos genéticos como soluções probabilísticas para a seleção das operações de escalonamento do JSSP. Tanto a solução por algoritmo genético padrão quanto pelo uso do NSGA-II resultaram em respostas satisfatórias para o problema, isto é, foram capazes de gerar uma saída ao problema do tipo *NP-hard*. Em uma análise numérica, o resultado obtido de *makespan* utilizando o NSGA-II demonstra ganho considerável em relação ao algoritmo genético padrão. Como trabalho futuro, destaca-se o estudo do JSSP como um problema de escalonamento com tempos de processamento e datas de entrega distintos para analisar o *Total Weighted Earliness and Tardiness* (TWET), configurando maior dificuldade ao problema.

Tabela 2: Configuração dos parâmetros de simulação.

Elemento	**Valor**	**Descrição**
Número de máquinas	10	-
Número de *jobs*	10	-
Quantidade de genes	100	Valor por cromossomo obtido pela multiplicação das quantidades de máquinas e *jobs*.
Tempo de processamento	-	Tabela que define o valor para cada uma das 10 operações dos 10 *jobs* [20].
Sequência de máquinas	-	Tabela que define a sequência de máquinas responsáveis por cada operação [20].
Tamanho da população	30	-
Taxa de cruzamento	0,8	O operador de cruzamento é aplicado para produzir populações de filhos distintas dos cromossomos pais.
Taxa de mutação	0,2	O operador de mutação é aplicado nos cromossomos para alteração dos genes.
Taxa de seleção de mutação	0,2	-
Quantidade de *jobs* mutados	20	-
Quantidade de iterações	2000	Configura a quantidade de laços do algoritmo e serve como critério de parada.

Figura 5: Valor de *makespan* ótimo em função das gerações para o algoritmo genético. Valor ótimo: 1180

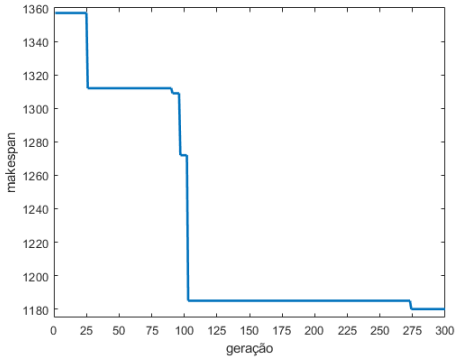


Figura 6: Valor de *makespan* ótimo em função das gerações para o NSGA-II. Valor ótimo: 996

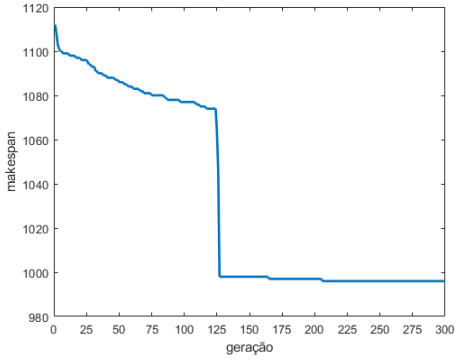
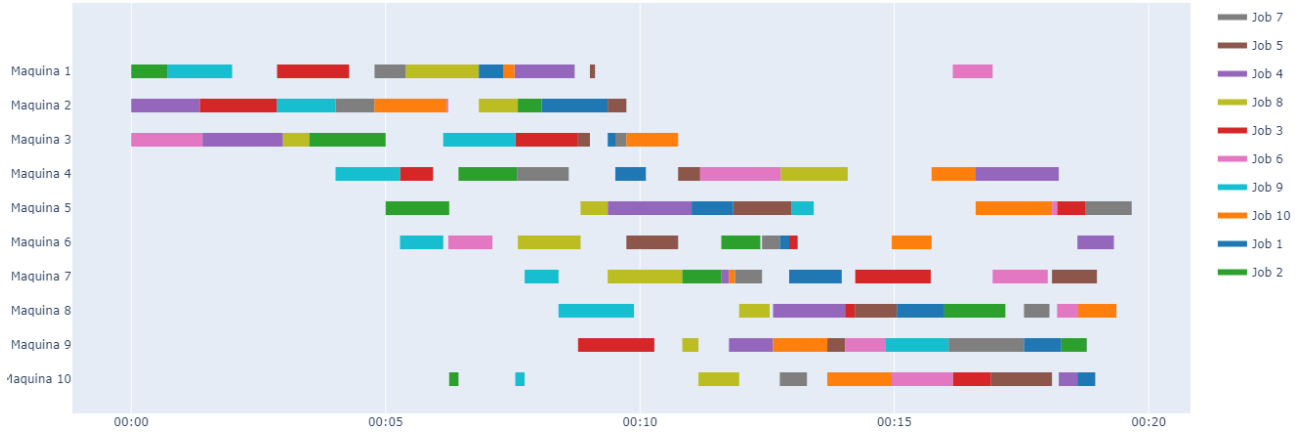


Figura 7: Diagrama de Grantt para o NSGA-II com *makespan* ótimo de 996.



AGRADECIMENTOS

Os autores agradecem o apoio financeiro concedido pelo CNPq (processo 131026/2022-4).

REFERÊNCIAS

[1] A. Mamane, M. Fattah, M. El Ghazi, M. El Bekkali, Y. Balboul, and S. Mazer, "*Scheduling algorithms for 5G networks and beyond: Classification and survey*," *IEEE Access*, vol. 10, pp. 51643-51661, 2022.

[2] A. M. Almeida, J. A. de Macêdo, and J. C. Machado, "*Optimization of Urban Semaphore Times Turning into JSSP*," in *BiDu-Posters@ VLDB*, 2018.

[3] R. A. Liaqait, S. Hamid, S. S. Warsi, and A. Khalid, "*A critical analysis of job shop scheduling in context of industry 4.0*," *Sustainability*, vol. 13, no. 14, p. 7684, 2021.

[4] S. Baer, J. Bakakeu, R. Meyes, and T. Meisen, "*Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems*," in *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*, 2019: IEEE, pp. 22-25.

[5] K. R. Baker and D. Trietsch, *Principles of sequencing and scheduling*. John Wiley & Sons, 2013.

[6] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, "*Sequencing and scheduling: Algorithms and complexity*," *Handbooks in operations research and management science*, vol. 4, pp. 445-522, 1993.

[7] N. Sadeh, *Micro-opportunistic scheduling: The micro-boss factory scheduler*. Citeseer, 1994.

[8] D. McDermott and J. Hendler, "*Planning: What it is, What it could be, An introduction to the Special Issue on Planning and Scheduling*," *Artificial Intelligence*, vol. 76, no. 1-2, 1995.

[9] R. Cheng, M. Gen, and Y. Tsujimura, "*A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation*," *Computers & industrial engineering*, vol. 30, no. 4, pp. 983-997, 1996.

[10] F. A. Rodammer and K. P. White, "*A recent survey of production scheduling*," *IEEE transactions on systems, man, and cybernetics*, vol. 18, no. 6, pp. 841-851, 1988.

[11] M. R. Garey, D. S. Johnson, and R. Sethi, "*The complexity of flowshop and jobshop scheduling*," *Mathematics of operations research*, vol. 1, no. 2, pp. 117-129, 1976.

[12] J. K. Lenstra and A. R. Kan, "*Computational complexity of discrete optimization problems*," in *Annals of discrete mathematics*, vol. 4: Elsevier, 1979, pp. 121-140.

[13] R. Nakano and T. Yamada, "*Conventional genetic algorithm for job shop problems*," in *ICGA*, 1991, vol. 91, pp. 474-479.

[14] M. Gen, Y. Tsujimura, and E. Kubota, "*Solving job-shop scheduling problems by genetic algorithm*," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, 1994, vol. 2: IEEE, pp. 1577-1582.

[15] T. Yamada and R. Nakano, "*Job shop scheduling*," *IEE control Engineering series*, pp. 134-134, 1997.

[16] S. Verma, M. Pant, and V. Snasel, "*A comprehensive review on NSGA-II for multi-ojective combinatorial optimization problems*," *Ieee Access*, vol. 9, pp. 57757-57791, 2021.

[17] A. Ben-Tal, "*Characterization of Pareto and lexicographic optimal solutions*," in *Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979*, 1980: Springer, pp. 1-11.

[18] D. A. Van Veldhuizen and G. B. Lamont, "*Evolutionary computation and convergence to a pareto front*," in *Late breaking papers at the genetic programming 1998 conference*, 1998: Citeseer, pp.221-228.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "*A fast and elitist multiobjective genetic algorithm: NSGA-II*," IEEE transactions on evolutionary computation, vol. 6, no. 2, pp. 182-197, 2002

[20] Wu. Min-You, "*Multi-Objective Stochastic Scheduling Optimization: A Study of Auto Parts Manufacturer in Taiwan*", Dissertação de mestrado, *Institute of Manufacturing Information and Systems, National Cheng Kung University*, 2016. Acedido em 3 de agosto de 2023, em: https://hdl.handle.net/11296/4e23b3. .