UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

GUSTAVO ANDRÉ CAUZZI

ALGORITMO GENÉTICO PARALELO PARA O ESCALONAMENTO DE TAREFAS

CAXIAS DO SUL

2025

GUSTAVO ANDRÉ CAUZZI

ALGORITMO GENÉTICO PARALELO PARA O ESCALONAMENTO DE TAREFAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Cientista da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. André Luis Martinotto

CAXIAS DO SUL

2025

GUSTAVO ANDRÉ CAUZZI

ALGORITMO GENÉTICO PARALELO PARA O ESCALONAMENTO DE TAREFAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Cientista da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado(a) em 26/11/2025

BANCA EXAMINADORA

Prof. Dr. André Luis Martinotto
Universidade de Caxias do Sul - UCS

Prof. Dr. Leonardo Dagnino Chiwiacowsky
Universidade de Caxias do Sul - UCS

Prof. Dra. Maria de Fátima Webber do Prado Lima
Universidade de Caxias do Sul - UCS

RESUMO

Problemas de escalonamento de tarefas estão presentes em diversas aplicações e a solução destes é fundamental para uma otimização no uso dos recursos. Entre os modelos existentes, destaca-se o problema de escalonamento do tipo *Flow Shop* permutacional, que possuem uma complexidade combinatória e cuja a solução por métodos exatos é limitada a instâncias de pequeno porte. Neste contexto, os algoritmos genéticos surgem como uma alternativa, devido à sua capacidade de explorar amplos espaços de busca e encontrar soluções de alta qualidade. No entanto, algoritmos genéticos podem demandar um alto custo computacional, particularmente em problemas que envolvem populações com grande número de indivíduos por geração. Portanto, neste trabalho foi desenvolvido uma implementação de algoritmos genéticos paralelos para a resolução do problema de escalonamento do tipo *Flow-Shop* permutacional, com o objetivo de reduzir o tempo total de execução das tarefas, isto é, o tempo necessário para que todas as tarefas sejam processadas em todas as máquinas do ambiente *Flow-Shop*. Para tanto, foi testado um modelo em ilhas bidimensionais, avaliando diferentes números de ilhas e número de gerações entre migrações. As implementações foram desenvolvidas para uma arquitetura de memória distribuída, utilizando um *cluster* de computadores, e a comunicação entre os processos foi realizada por meio da biblioteca *Message Passing Interface* (MPI). A qualidade das soluções foi avaliada a partir da base de testes proposta por Taillard (1993), amplamente utilizada na literatura. Os resultados obtidos mostram que tanto a versão sequencial do algoritmo genético quanto a versão paralela em modelo de ilhas apresentaram resultados superiores aos reportados na literatura. Por fim, nos testes limitando o número de avaliações da função objetivo, a versão paralela alcançou um *speedup* de cerca de 10 vezes em um computador com 10 núcleos, indicando que o custo de comunicação da migração de indivíduos não afetou significativamente o desempenho da implementação paralela.

Palavras-chave: Escalonamento de Tarefas. *Flow Shop* permutacional. Algoritmos Genéticos paralelos. *Clusters* de Computadores.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ENMA	Elite, Novidade, Mutado, Aleatório
MPI	*Message Passing Interface*
SPMD	*Single Program, Multiple Data*
Pthreads	POSIX *Threads*
OpenMP	*Open specifications for Multi Processing*
PVM	*Parallel Virtual Machine*
OpenMPI	*Open Message Passing Interface*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
NASA	*National Aeronautics and Space Administration*
UB	*Upper Bound*
GHz	*Gigahertz*
GB	*Gigabyte*
MB	*Megabyte*
KB	*Kilobyte*
SSD	*Solid State Drive*
NVMe	*Non-Volatile Memory Express*
LTS	*Long Term Support*

SUMÁRIO

1 INTRODUÇÃO

O problema de escalonamento de tarefas é recorrente em diversas áreas, sendo especialmente relevante no contexto do paradigma de computação em nuvem (SINGH; SINGH; VARSNEY, 2024). Essa classe de problemas consiste em identificar a melhor permutação de tarefas em um conjunto fixo de recursos, visando otimizar algum critério pré-definido, como, por exemplo, a minimização do tempo total de execução (XIONG *et al.*, 2022).

No contexto da computação em nuvem, o problema de escalonamento pode ser modelado como a distribuição de tarefas entre instâncias de máquinas virtuais (recursos), com o objetivo de reduzir o tempo total de execução. Essas tarefas podem representar, por exemplo, unidades de processamento de dados, execuções de microsserviços ou tarefas de treinamento de modelos de inteligência artificial (CALHEIROS *et al.*, 2011; YOUSEFF; BUTRICO; SILVA, 2008). Um escalonamento eficiente de tarefas é essencial para garantir uma alocação otimizada dos recursos (SONG; GAO; WANG, 2011).

O problema de escalonamento de tarefas é classificado como NP-difícil[1], pois não existe um algoritmo conhecido capaz de encontrar uma solução ótima em tempo polinomial (HANEN, 1994). Isso implica que o tempo necessário para computar a solução ótima cresce exponencialmente com o aumento no número de tarefas e/ou recursos. Desta forma, encontrar a solução ótima torna-se impraticável para instâncias de grande tamanho, devido ao elevado custo computacional (WOEGINGER, 2003).

Uma alternativa para contornar esse problema é a utilização de heurísticas. Essas técnicas exploram progressivamente o espaço de soluções de um problema. Inicialmente, é aplicada uma heurística construtiva, que constrói uma solução inicial de forma incremental. Em seguida, essa solução é refinada por meio de uma heurística de melhoramento local, na qual são gerados vizinhos a partir de pequenas modificações no escalonamento corrente (ALBA, 2005). Essas técnicas não possuem garantia de que a solução ótima seja encontrada (YANG, 2008). Porém, a ideia principal é ter um algoritmo rápido o suficiente e que produza soluções satisfatórias, podendo algumas vezes aproximar-se da solução ótima (GANDOMI *et al.*, 2013).

Além disso, podem ser utilizadas metaheurísticas para refinar as técnicas de busca, permitindo assim uma exploração mais eficiente do espaço de soluções e aumentando a probabilidade de se encontrar a solução ótima. Essas metaheurísticas incluem os algoritmos de colônia de formigas, pesquisa local iterada, busca tabu e algoritmos genéticos (GANDOMI *et al.*, 2013). A escolha da metaheurística dependerá do objetivo e das características do problema a ser re-

[1] Problemas NP-difíceis pertencem a uma classe cujas soluções não podem ser encontradas, nem verificadas, em tempo polinomial. Essa classe está frequentemente associada a problemas de otimização, nos quais a solução não é simplesmente uma decisão binária (por exemplo, "Sim"ou "Não"), mas sim a busca por uma solução ótima dentro de um grande espaço de possibilidades. (GAREY; JOHNSON, 1979; PAPADIMITRIOU; STEIGLITZ, 1998).

solvido (GÖKALP; UĞUR, 2020).

Neste sentido, nota-se uma ampla adoção de algoritmos genéticos em pesquisas sobre escalonamento de tarefas. De fato, esses algoritmos têm sido amplamente utilizados devido aos resultados promissores em problemas complexos de otimização (SASAKI *et al.*, 2010; SISSODIA; RAUTHAN; BARTHWAL, 2022; MANAVI; ZHANG; CHEN, 2023; LORENA; LOPES, 1997; EBRAHIMI *et al.*, 2023; REEVES, 1995). Porém, dependendo da natureza do problema, o tempo requerido para a convergência de um algoritmo genético pode ser muito elevado (ZHANG; WANG, 2019). Para solucionar esse problema, podem ser desenvolvidas implementações paralelas desses algoritmos, sendo que as estratégias de paralelização mais comuns são os algoritmos genéticos distribuídos globais e algoritmos genéticos distribuídos em ilhas (HARADA; ALBA, 2020; COHOON; PARIS, 1987).

Neste trabalho foi desenvolvida uma implementação paralela de um algoritmo genético para a solução do problema de escalonamento de tarefas. A implementação paralela foi desenvolvida utilizando um modelo em ilhas bidimensionais, avaliando-se o número de ilhas e o número de gerações entre migrações. A implementação foi desenvolvida para ser executada em *clusters* de computadores, pois esses são uma alternativa relativamente barata e simples para lidar com problemas que exigem um alto poder computacional (BUYYA *et al.*, 2009). Para o desenvolvimento foi utilizado o padrão de troca de mensagens MPI (*Message Passing Interface*), amplamente utilizado pela comunidade de programação paralela (GROPP; SNIR, 1998).

1.1 OBJETIVOS

Este trabalho teve como principal objetivo desenvolver uma implementação paralela de um algoritmo genético para resolver o problema de escalonamento de tarefas. A finalidade desse algoritmo é minimizar o tempo total de execução, isto é, o tempo necessário para que todas as tarefas sejam processadas em todas as máquinas do ambiente *Flow-Shop*, em um ambiente de computação em nuvem sob demanda. Para atingir esse objetivo, os seguintes objetivos específicos foram realizados:

1. Modelar um algoritmo genético para resolver o problema de escalonamento de tarefas;
2. Implementar uma versão sequencial do algoritmo genético;
3. Desenvolver uma versão paralela do algoritmo genético;
4. Avaliar o desempenho da implementação paralela considerando tempo de execução, *speedup* e eficiência.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado da seguinte forma. Neste capítulo é apresentada uma breve descrição do problema, bem como a motivação e os objetivos do estudo. No Capítulo 2 é introduzido o problema de escalonamento de tarefas e são apresentados os trabalhos relacionados. No Capítulo 3 são descritos os principais conceitos de algoritmos genéticos, bem como os operadores genéticos utilizados. No Capítulo 4 são discutidos os principais conceitos de programação paralela, com ênfase em *clusters* de computadores, arquitetura adotada neste trabalho. O Capítulo 5 apresenta a implementação desenvolvida, contemplando a versão sequencial do algoritmo genético e o modelo em ilhas adotado. No Capítulo 6 são apresentados e discutidos os resultados obtidos. Por fim, no Capítulo 7 são apresentadas as considerações finais do trabalho e as sugestões de trabalhos futuros.

2 ESCALONAMENTO DE TAREFAS

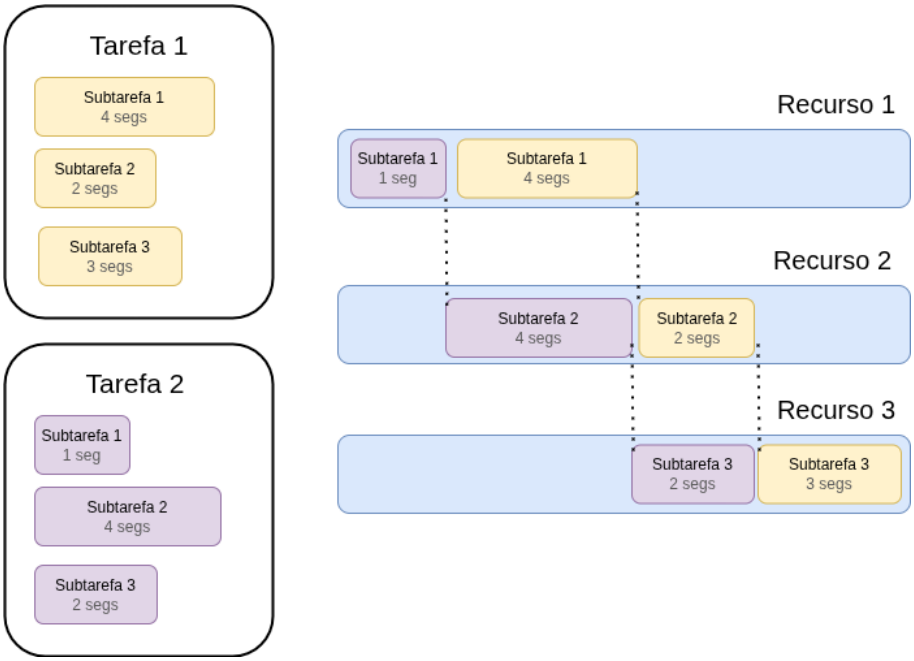
O problema de escalonamento de tarefas é uma classe de problemas de otimização com utilização em diversas áreas, incluindo a manufatura, logística, hospitalar, entre outras. Por exemplo, na manufatura, a otimização pode ser utilizada em uma linha de produção de forma a maximizar a quantidade de itens produzidos (PARVEEN; ULLAH, 2011). Em hospitais, a otimização pode ser utilizada no agendamento de consultas e cirurgias, sendo essencial para melhorar a alocação de salas e aumentar a eficiência operacional (AZAIEZ *et al.*, 2022).

Na área da ciência da computação, uma das principais aplicações do escalonamento de tarefas está no paradigma de computação em nuvem sob demanda (SINGH; SINGH; VARSNEY, 2024). Neste caso, uma alocação eficiente dos recursos computacionais é essencial para garantir um maior desempenho, escalabilidade e redução de custos (MUCHALSKI, 2014; SONG; GAO; WANG, 2011). De fato, um escalonamento ineficiente pode levar a uma sobrecarga nos servidores, gerando o desperdício de recursos e atraso nas execuções das tarefas (KUMAR *et al.*, 2019).

O objetivo do escalonamento de tarefas consiste em minimizar algum critério de desempenho previamente definido, como por exemplo, a redução do atraso das tarefas em relação aos prazos, o balanceamento da carga entre as máquinas ou, como abordado neste trabalho, a minimização do tempo total de execução (WANG; SUN; SUN, 2015). Esse consiste em alocar um conjunto de tarefas em um número fixo de máquinas, definindo uma ordem de execução e respeitando as restrições do problema. Neste caso, cada tarefa pode ser composta por subtarefas com durações diferentes, sendo que cada subtarefa deve ser processada em uma máquina específica por um tempo ininterrupto (XIONG *et al.*, 2022). Além disso, as subtarefas podem apresentar restrições de precedência, garantindo que uma subtarefa só poderá ser executada se todas as suas predecessoras tiverem sido concluídas (XIONG *et al.*, 2022; M.; KUMAR; R, 2022). A maneira como essas restrições e dependência estão estruturadas determinará o tipo do problema de escalonamento (PINEDO, 2016).

A Figura 1 ilustra uma instância do problema de escalonamento de tarefas, cujo objetivo é minimizar o tempo total de execução. O exemplo apresenta duas tarefas e três máquinas, onde cada uma das tarefas é formada por três subtarefas, com restrições de precedência entre as mesmas. Neste caso, existem apenas duas permutações possíveis, sendo que o melhor escalonamento é aquele em que a *Tarefa 2* é executada antes da *Tarefa 1*, resultando em um tempo total de execução de 10 segundos. A ordem inversa resultaria em um tempo total de execução de 12 segundos.

Figura 1 – Escalonamento de duas tarefas em três máquinas.



Fonte: O Autor

2.1 CLASSIFICAÇÃO DOS PROBLEMAS DE ESCALONAMENTO

O problema de escalonamento consiste na atribuição de um conjunto de recursos para a execução de uma coleção de tarefas, sendo que a categorização do problema depende das restrições envolvidas (PARVEEN; ULLAH, 2011). A categorização é essencial para uma correta modelagem do problema e para a escolha das técnicas de solução, visto que cada classe possui restrições que afetam diretamente na complexidade do problema (PINEDO, 2016). Geralmente, os problemas de escalonamento são divididos em duas categorias: *Flow-Shop* e *Job-Shop* (PARVEEN; ULLAH, 2011; PESCH, 1994; ŠEDA, 2007).

2.1.1 Modelo *Flow-Shop*

O modelo *Flow-shop* é composto por um conjunto de tarefas e um conjunto de máquinas. Neste caso, cada tarefa é dividida em um número fixo de subtarefas correspondente à quantidade de máquinas disponíveis. Posteriormente, cada subtarefa é atribuída a sua respectiva máquina. Por exemplo, as *Subtarefas 1* são atribuídas a *Máquina 1*, as *Subtarefas 2* são atribuídas a *Máquina 2*, e assim sucessivamente até que todas as subtarefas tenham sido concluídas (DAVIS, 1991; WANG; SUN; SUN, 2015).

O modelo *Flow-shop* pode ser dividido em: o *Flow-shop* permutacional e o *Flow-shop* não permutacional. Esses modelos diferem quanto às restrições impostas à ordem de execução das subtarefas (WANG; SUN; SUN, 2015). No modelo *Flow-shop* permutacional, a ordem de execução das subtarefas deve ser preservada em todas as máquinas. Por exemplo, se a primeira

subtarefa da *Tarefa 1* for executada antes da primeira subtarefa da *Tarefa 2* na *Máquina 1*, então a segunda subtarefa da *Tarefa 1* também deverá ser executada antes da segunda subtarefa da *Tarefa 2* na *Máquina 2*, e assim sucessivamente nas demais máquinas. Desta forma, a definição do escalonamento se reduz à escolha de uma única permutação das tarefas, que será aplicada de forma idêntica em todas as máquinas do sistema (DAVIS, 1991; ŠEDA, 2007; PARVEEN; ULLAH, 2011).

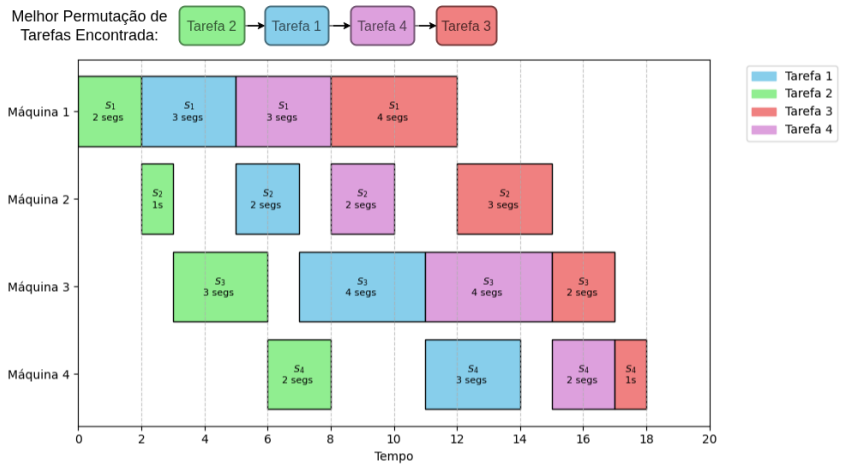
A Figura 2 ilustra a estrutura de um problema *Flow-Shop* permutacional com quatro tarefas e quatro máquinas. O objetivo consiste em determinar a melhor permutação das tarefas, de forma a minimizar o tempo total de processamento.

Figura 2 – Modelo de *Flow-shop* permutacional.

Fonte: O Autor

A Figura 3 apresenta a solução ótima, na qual as tarefas são executadas na ordem *Tarefa 2* $\rightarrow$ *Tarefa 1* $\rightarrow$ *Tarefa 4* $\rightarrow$ *Tarefa 3*. Observa-se que todas as subtarefas são executadas nesta ordem em cada uma das máquinas, obtendo-se um tempo total de execução de 18 segundos.

Figura 3 – Solução ótima ao problema de *Flow-shop* permutacional

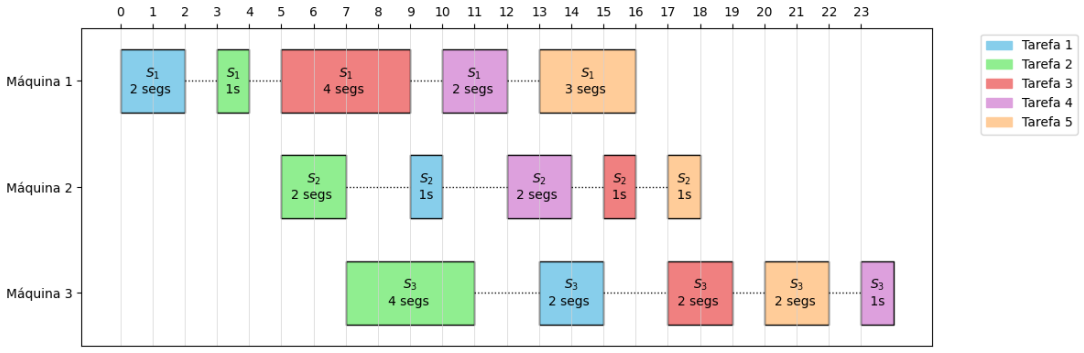


Fonte: O Autor

Por outro lado, no modelo *Flow-shop* não permutacional, não existe nenhuma restrição quanto à ordem de execução das tarefas nas máquinas. Por exemplo, mesmo que a primeira subtarefa da *Tarefa 1* seja processada antes da segunda subtarefa da *Tarefa 2* na *Máquina 1*, nada impede que a segunda subtarefa da *Tarefa 2* seja processada antes da segunda subtarefa da *Tarefa 1* na *Máquina 2*. O modelo não permutacional é, portanto, uma generalização do modelo permutacional. Todo escalonamento permutacional é também um escalonamento válido no modelo não permutacional, mas o inverso não é verdadeiro.

A Figura 4 apresenta uma solução viável para um problema de *Flow-shop* não permutacional com cinco tarefas. No escalonamento apresentado, a *Máquina 1* processa as tarefas na ordem 1, 2, 3, 4 e 5; já a *Máquina 2* segue a ordem 2, 1, 4, 3 e 5; enquanto a *Máquina 3* processa as tarefas nas ordens 2, 1, 3, 5 e 4, respectivamente. Essa flexibilidade expande o espaço de soluções em comparação ao caso permutacional, onde a ordem de execução deve ser idêntica em todas as máquinas (RITT; ROSSIT, 2024).

Figura 4 – Possível solução para um problema de *Flow-shop* não permutacional



Fonte: Adaptado de Ferreira, Faria e Alves (2021)

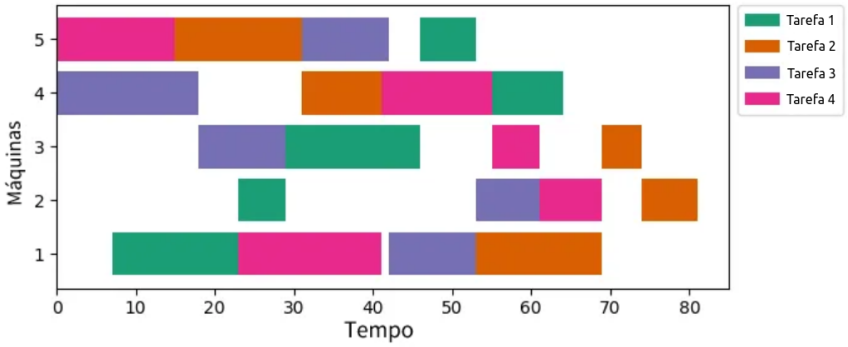
O modelo *Flow-shop* permutacional, mesmo sendo mais simples, é o mais estudado devido a sua maior aplicabilidade em situações reais (REEVES, 1995; XIAO *et al.*, 2012). O modelo *Flow-shop* não permutacional é menos representativo em cenários reais, sendo por isso menos explorado na literatura (PINEDO, 2016). Neste trabalho optou-se por estudar o problema de *Flow-shop* permutacional devido à sua maior aplicabilidade (DAVIS, 1991; PINEDO, 2016).

2.1.2 Modelo *Job-Shop*

O modelo *Job-Shop* é mais flexível que o modelo *Flow-Shop*. Nesse modelo, cada tarefa pode ter suas subtarefas processadas em uma sequência diferente de máquinas. Isso permite que, por exemplo, uma tarefa inicie seu processamento na *Máquina 1*, enquanto outra inicie na *Máquina 5*, e assim por diante. Além disso, cada tarefa pode conter um número variável de subtarefas, diferentemente do *Flow-Shop*, onde toda a tarefa apresenta um número de subtarefas iguais ao número de máquinas (DAUZèRE-PéRèS *et al.*, 2024).

A Figura 5 ilustra uma solução possível para um problema *Job-Shop* com quatro tarefas e cinco máquinas. Neste caso, observa-se que a ordem de execução das subtarefas de uma tarefa variam entre as máquinas. Por exemplo, a *Tarefa 1* inicia na *Máquina 1*, passa pela *Máquina 5* e é finalizada na *Máquina 4*. Já a *Tarefa 4* segue uma sequência completamente distinta. Essa flexibilidade aumenta a complexidade do problema, tanto do ponto de vista da modelagem quanto de solução (DAUZèRE-PéRèS *et al.*, 2024).

Figura 5 – Solução para um problema *Job-Shop*



Fonte: Adaptado de (LEITE, 2023)

2.2 COMPLEXIDADE DOS ALGORITMOS DE ESCALONAMENTO

Para a obtenção da solução ótima, é necessária, a avaliação de todas as possíveis sequências de execução, tornando o problema computacionalmente inviável para instâncias de maior tamanho (BLAZEWICZ *et al.*, 2013). Desta forma, algoritmos de força bruta que exploram todas as possíveis soluções são viáveis somente para instâncias pequenas. De fato, à medida que o número de máquinas ou tarefas aumenta, o tempo de computação se torna impraticável, devido à natureza combinatória do problema (GAREY; JOHNSON; SETHI, 1976).

Os problemas categorizados como *Flow-shop* podem ser resolvidos em tempo polinomial somente em instâncias envolvendo duas máquinas e qualquer número de tarefas utilizando o algoritmo de Johnson (JOHNSON, 1954). Porém, para três ou mais máquinas, a utilização de força bruta para exploração de todas as combinações classifica o problema como NP-completo[1] na sua forma de decisão[2] e NP-difícil[3] na forma de otimização[4] (GAREY; JOHNSON; SETHI, 1976). No problema de *Flow-shop* de permutação, em sua forma de otimização, o número de possíveis sequências de execução cresce de forma fatorial com o número de tarefas (t), resultando em uma complexidade de $O(t!)$. Para a subclasse de *Flow-shop* sem permutação, como cada máquina m pode ter uma sequência independente das demais, a complexidade aumenta

para O($t!m$) (GAREY; JOHNSON; SETHI, 1976).

O problema de escalonamento do tipo *Job-shop* na sua forma de otimização, também é classificado como NP-Difícil (GAREY; JOHNSON; SETHI, 1976). Diferentemente do *Flow-shop*, o problema *Job-shop* não admite solução em tempo polinomial mesmo quando restrito a duas máquinas. No entanto, quando o número de tarefas é limitado a duas, o problema pode ser resolvido de maneira polinomial, independentemente do número de máquinas (BAKER, 2014).

Uma alternativa à verificação de todas as combinações possíveis de execução é a utilização de algoritmos heurísticos, como por exemplo, os métodos gulosos (PAN; WANG; ZHAO, 2007). Esses constroem as soluções de forma sequencial a partir de uma solução inicialmente vazia. Essa solução é incrementada com base em decisões a partir do estado atual da solução. Cada solução pode ser verificada por meio de uma função de avaliação, permitindo verificar se a mesma está progredindo ou regredindo em relação ao objetivo final (PAN; WANG; ZHAO, 2007). Essas técnicas são altamente suscetíveis a ficarem presas em ótimos locais, dificultando a obtenção da solução ótima global à medida que o espaço de busca aumenta (ALBA, 2005). A Figura 6 apresenta um possível espaço de solução, mostrando a representação de um ótimo local e de um ótimo global em um problema de maximazão.

Figura 6 – Ótimos locais e global em um espaço de soluções.

Fonte: Adaptado de (MATSUMURA, 2020)

[1] Problemas NP-completos são aqueles que pertencem à classe NP, ou seja, se fornecida uma solução candidata, é possível verificar sua validade em tempo polinomial, e que também são NP-difíceis. Isto é, qualquer problema da classe NP pode ser reduzido a eles em tempo polinomial. Essa dupla condição os torna os mais difíceis dentro de NP. Se algum problema NP-completo fosse resolvido em tempo polinomial, todos os problemas em NP também poderiam ser resolvidos nesse tempo. A principal diferença para os problemas NP-difíceis é que estes não precisam estar em NP, e portanto podem não ter verificação polinomial (GAREY; JOHNSON, 1979).

[2] Um problema de decisão é definido como aquele que busca uma resposta binária para uma determinada condição. No contexto, *Flow-Shop* de decisão é aquele que, por exemplo, onde se quer saber se existe um tempo mínimo de execução menor que um determinado tempo (GAREY; JOHNSON, 1979).

[4] Problemas de otimização visam procurar pela melhor solução possível de acordo com algum critério. Diferentemente dos problemas de decisão, sua resposta não é binária. Um exemplo dessa classe é a busca pela sequência de tarefas no *Flow-shop* que minimizam o tempo total de execução (PINEDO, 2016).

Diante dessa limitação, diversas abordagens baseadas em metaheurísticas têm sido exploradas na literatura. Entre essas metaheurísticas destacam-se os algoritmos genéticos, o recozimento simulado, a busca tabu, a otimização por colônia de formigas e a otimização por enxame de partículas (ALBA, 2005). A capacidade dessas técnicas de fornecerem soluções quase ótimas em tempos computacionais razoáveis as tornaram escolhas preferenciais para uma ampla variedade de problemas de combinação, incluindo o escalonamento de tarefas (PINEDO, 2016).

2.3 TRABALHOS RELACIONADOS

Esta seção apresenta trabalhos que comparam diferentes metaheurísticas clássicas, como recozimento simulado, busca tabu e algoritmos genéticos, em problemas de alocação de recursos e otimização combinatória. O objetivo é evidenciar o desempenho relativo dessas abordagens e, em particular, destacar a capacidade dos algoritmos genéticos de produzir soluções de boa qualidade em instâncias de maior porte, o que fundamenta sua escolha como foco deste trabalho.

Connor e Shah (2014) realizaram uma comparação entre as metaheurísticas de recozimento simulado, busca tabu e algoritmos genéticos na resolução do problema de alocação de recursos em sistemas distribuídos. O estudo avaliou a eficiência de cada abordagem considerando tanto a qualidade das soluções obtidas quanto a escalabilidade frente ao aumento no número de tarefas e recursos. Os resultados indicaram que os algoritmos genéticos apresentaram desempenho superior, conseguindo encontrar soluções de menor custo e maior eficiência de alocação, especialmente em cenários com um maior número de recursos. De fato, embora as metaheurísticas de busca tabu e recozimento simulado tenham obtido desempenhos satisfatórios para problemas de menor porte, ambos perderam eficiência à medida que a escala do problema crescia.

De forma similar, Said, Mahmoud e El-Horbaty (2014) realizaram uma comparação das mesmas metaheurísticas na solução do problema quadrático de alocação [1], um clássico problema NP-difícil. Os resultados obtidos mostraram que os algoritmos genéticos apresentaram desempenho superior, especialmente em instâncias de maior porte. A busca tabu demonstrou bom desempenho em instâncias intermediárias, embora dependesse de um ajuste fino dos parâmetros. Já o recozimento simulado se mostrou eficiente apenas para instâncias de pequeno porte, perdendo a eficiência conforme o aumento do tamanho das instâncias.

No trabalho desenvolvido por Silva *et al.* (2005), foram comparadas as metaheurísticas de algoritmos genéticos e de colônia de formigas na resolução de problemas de otimização combinatória. Os resultados indicaram que os algoritmos genéticos, embora exigissem maior

[1] O problema quadrático de alocação é um problema clássico de otimização combinatória, classificado como NP-difícil. Ele consiste em alocar um conjunto de n unidades (como tarefas, departamentos ou equipamentos) a n locais, minimizando o custo total associado à interação entre as unidades e à distância entre os locais. A função objetivo envolve termos quadráticos, pois o custo depende de pares de alocações simultaneamente (BURKARD; ÇELA, 1998).

tempo computacional, obtiveram soluções de qualidade superior em comparação à colônia de formigas. Por outro lado, o algoritmo de colônia de formigas demonstrou tempos de convergência mais rápidos, mas com tendência a encontrar soluções subótimas em problemas com instâncias maiores.

No trabalho desenvolvido por Karthick e Ramaraj (2023), foi realizada uma análise comparativa entre as metaheurísticas de algoritmos genéticos, colônia de formigas e otimização por enxame de partículas. Essas técnicas foram aplicadas no contexto de escalonamento de tarefas em computação em nuvem. O estudo avaliou a eficiência dos algoritmos com base no tempo de execução das tarefas, na utilização de recursos e na capacidade de atender às tarefas. Os resultados apontaram que tanto os algoritmos genéticos quanto a otimização por enxame de partículas foram capazes de gerar soluções satisfatórias, apresentando melhor desempenho em comparação à colônia de formigas. Os autores concluem que ambas as abordagens são adequadas para lidar com problemas de agendamento de tarefas em ambientes de nuvem.

No trabalho de Reeves (1995), foram utilizados algoritmos genéticos no problema de escalonamento *Flow-shop* com permutação. Foram realizadas comparações entre algoritmos genéticos e outras metaheurísticas, como o recozimento simulado e uma busca local com vizinhança simples. Para comparações, foram utilizadas instâncias geradas artificialmente, bem como bases de dados da literatura, como os conjuntos propostos por Taillard (1993). Os resultados mostram que o algoritmo genético gerou resultados ligeiramente superiores, em média, ao algoritmo de recozimento simulado e busca local com vizinhança simples.

2.4 ESCOLHA DA METAHEURÍSTICA

Devido ao melhor desempenho observado em estudos comparativos entre metaheurísticas, optou-se pela utilização de algoritmos genéticos para a solução do problema de escalonamento de tarefas *Flow-Shop* permutacional. A literatura mostra uma ampla uitlização de algoritmos genéticos na resolução de problemas de otimização, resolvendo problemas matemáticos (LORENA; LOPES, 1997), problems de localização geográfica de ambulatórios (SASAKI *et al.*, 2010), planejamento de rotas (EBRAHIMI *et al.*, 2023), entre outros. Da mesma forma, destacam-se aplicações específicas em problemas de escalonamento, incluindo o escalonamento do tipo *Flow-shop* (REEVES, 1995; SISSODIA; RAUTHAN; BARTHWAL, 2022; MANAVI; ZHANG; CHEN, 2023; EBRAHIMI *et al.*, 2023), nos quais os algoritmos genéticos apresentaram um bom desempenho ao explorar grandes espaços combinatórios de soluções.

A superioridade dos algoritmos genéticos em problemas combinatórios de alta complexidade pode ser atribuída a suas características estruturais. Por operarem sobre uma população de soluções, em vez de uma única solução que executa uma busca, os algoritmos genéticos conseguem explorar múltiplas regiões do espaço de busca simultaneamente, aumentando as chances de encontrar regiões promissoras, possivelmente uma delas próxima do ótimo global (ALBA, 2005; CONNOR; SHAH, 2014). Além disso, os operadores de seleção, cruzamento e mutação também contribuem para um balanço adequado entre diversificação e intensificação (SAID; MAHMOUD; EL-HORBATY, 2014; SILVA *et al.*, 2005).

3 ALGORITMOS GENÉTICOS

Algoritmos genéticos têm se mostrado particularmente eficazes na solução de problemas de otimização combinatória, graças à sua flexibilidade e capacidade de exploração de grandes espaços de busca (ALBA, 2005). De fato, nos últimos anos, esses têm sido utilizados em diversas áreas do conhecimento, incluindo engenharia (CASSAR; SANTOS; ZANOTTO, 2020), logística (MALASHIN *et al.*, 2024), bioinformática (BOONE *et al.*, 2021), saúde (HUANG; YIN; HUANG, 2024), ciência de dados (ISLAM; RAFA; KIBRIA, 2021), entre outras.

Os algoritmos genéticos foram inspirados na teoria da evolução de Darwin, transformando conceitos como a seleção natural e sobrevivência dos mais aptos em mecanismos computacionais aplicáveis a uma ampla gama de problemas (KATOCH; CHAUHAN; KUMAR, 2021; CARR, 2014). Esses foram inicialmente propostos por John Holland, na década de 1970, com o objetivo de modelar computacionalmente os mecanismos de adaptação observados na natureza (CARR, 2014). Eles baseiam-se nos princípios da seleção natural e da genética populacional, simulando um processo evolutivo que busca, ao longo de várias gerações, refinar uma população de soluções (HOLLAND, 1992).

Nos algoritmos genéticos, as soluções para um dado problema são representadas por cromossomos, os quais são modificados por meio de operadores inspirados em processos biológicos, como seleção, cruzamento e mutação. Ao aplicar iterativamente esses operadores, espera-se que a população convirja para regiões do espaço de busca com soluções de maior qualidade. Cada ciclo completo de um algoritmo genético, denominado geração, envolve as etapas de avaliação, seleção, cruzamento e mutação. Ao final de cada geração, a população é atualizada, incorporando os filhos gerados. Assim, ao longo de sucessivas gerações, espera-se que as soluções evoluam em direção a configurações mais eficientes (ALBA, 2005).

3.1 CODIFICAÇÃO

Um cromossomo é uma estrutura de dados que representa uma possível solução para o problema de otimização. A forma de representação do cromossomo varia conforme o tipo de problema: cadeias de *bits* são amplamente utilizadas em problemas binários; vetores de números reais são adequados para domínios contínuos; e vetores de inteiros são a escolha mais comum em problemas como o escalonamento de tarefas. Neste caso, cada valor inteiro é chamado de gene (REEVES, 1995).

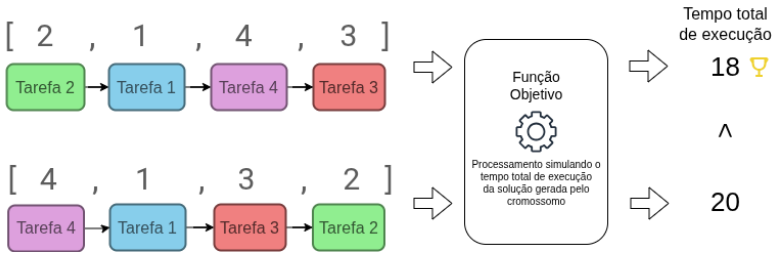
Em um problema de escalonamento de tarefas, cada cromossomo representa uma ordem de execução. Por exemplo, a sequência $[2, 1, 4, 3]$ indica que a *Tarefa 2* deve ser executada primeiro, seguida da *Tarefa 1*, e assim por diante. Essa estrutura é suficientemente adequada para representar uma solução de um problema do modelo *Flow-shop* permutacional, uma vez

que define de forma inequívoca a ordem de execução das tarefas (AKHSHABI; HADDADNIA; AKHSHABI, 2012).

3.2 FUNÇÃO OBJETIVO

Em algoritmos genéticos, a avaliação de cada solução é realizada por meio de uma função objetivo, que associa um valor numérico à qualidade da solução. Esse valor é calculado com base nas informações disponíveis em um cromossomo e permite a comparação entre as diferentes soluções (REEVES, 1995). No contexto do problema de escalonamento *Flow-Shop*, é comum adotar uma função objetivo que busca minimizar o tempo total de execução. Esse tempo corresponde ao intervalo necessário para que todas as tarefas sejam processadas. A Figura 7 apresenta dois cromossomos e o cálculo do tempo total de execução associado a cada um deles. O primeiro cromossomo, representado pela sequência das tarefas $[2, 1, 4, 3]$, possui um tempo total de execução igual a 18. Já o segundo cromossomo, $[4, 1, 3, 2]$, gera um tempo total de execução de 20.

Figura 7 – Função objetivo



Fonte: O Autor.

3.3 INICIALIZAÇÃO DA POPULAÇÃO

O primeiro passo de um algoritmo genético é a geração de uma população inicial. Essa população representa um conjunto de cromossomos, onde cada um representa uma possível solução. A qualidade e a diversidade dessa geração influencia diretamente na eficiência da busca (AWADALLAH; ELAZIZ, 2020). Neste trabalho, foi adotada a forma mais comum de inicialização da população que consiste na geração aleatória dos cromossomos. No caso do problema *Flow-shop*, isso significa criar diferentes permutações das tarefas. Essa abordagem é amplamente utilizada por sua facilidade de implementação e por garantir uma boa diversidade (LIU; ZHANG; WANG, 2023).

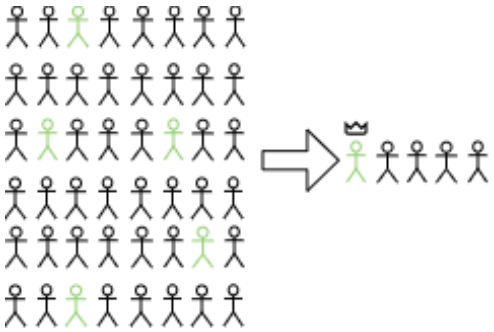
3.4 SELEÇÃO DOS PAIS

A seleção dos pais é a etapa responsável pela escolha dos indivíduos que irão participar do processo de cruzamento, originando os cromossomos da próxima geração. Essa escolha fre-

quentemente é baseada nos valores de aptidão das soluções, de modo que os indivíduos mais aptos tenham uma maior probabilidade de serem selecionados. O objetivo desse mecanismo é favorecer a propagação de características vantajosas, ao mesmo tempo de preserva uma diversidade populacional (HOLLAND, 1992). Diversos métodos de seleção são apresentados, sendo que os mais utilizados são seleção por roleta (LIU; WANG; ZHANG, 2016), seleção por torneio (MILLER; GOLDBERG, 1995) e seleção por *ranking* (BLICKLE; THIELE, 1995).

Neste trabalho, optou-se pela utilização da seleção por torneio. Nela, um pequeno subconjunto de candidatos é sorteado aleatoriamente a partir da população, e o indivíduo com melhor valor de aptidão dentro desse grupo é declarado vencedor e selecionado como genitor. O parâmetro central desse procedimento é o tamanho do torneio, onde torneios maiores aumentam a probabilidade de vitória de indivíduos de alta qualidade, acelerando a intensificação, ao passo que valores menores preservam oportunidades para candidatos medianos, contribuindo para a diversidade. Em versões mais gerais do método, pode-se empregar um parâmetro que define a probabilidade de o melhor indivíduo do torneio ser aceito, o que permite ajustar o equilíbrio entre intensificação e diversificação. No algoritmo desenvolvido neste trabalho, adotou-se a versão determinística, em que o melhor indivíduo do torneio é sempre selecionado (MILLER; GOLDBERG, 1995; EIBEN; SMITH, 2015). A Figura 8 ilustra um processo de torneio de cinco indivíduos, onde essa quantidade de cromossomos são selecionados aleatóriamente da população total, e o melhor é determinado como o progenitor do novo indivíduo.

Figura 8 – Exemplo de seleção por torneio.



Fonte: O Autor.

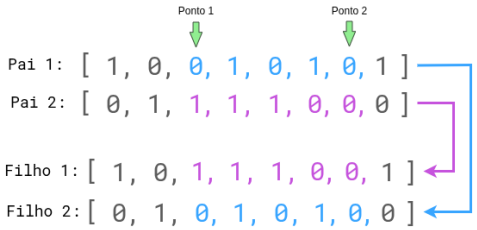
3.5 CRUZAMENTO

O cruzamento tem como objetivo gerar novos indivíduos a partir da combinação de dois ou mais indivíduos. Esse operador é responsável por promover a combinação de características, permitindo que boas soluções sejam propagadas e possivelmente melhoradas ao longo das gerações (ALBA, 2005).

Em problemas com representação binária ou vetorial, operadores de cruzamento simples, como os de um ponto ou de dois pontos, são frequentemente utilizados. Nesses casos, os genes dos pais são divididos em uma ou duas posições específicas, e partes de suas estruturas

são recombinadas de forma a gerar os descendentes (HOLLAND, 1992). A Figura 9 ilustra um cruzamento de dois pontos. Nesse exemplo, dois pontos são selecionados aleatoriamente, e os segmentos delimitados por esses pontos são trocados. O resultado é a geração de dois descendentes que combinam as partes de ambos os pais.

Figura 9 – Operação de Cruzamento

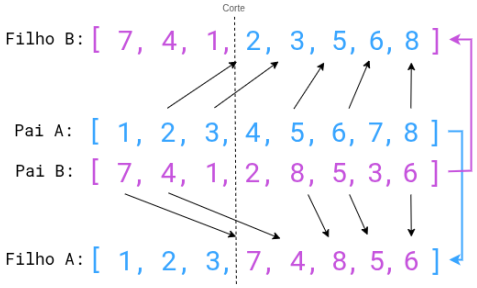


Fonte: O Autor.

Em problemas como o escalonamento do tipo *Flow-shop*, onde os cromossomos representam as permutações de tarefas, o uso desses operadores pode gerar soluções inválidas. De fato, a simples troca de segmentos entre cromossomos pode resultar em repetições ou omissões de tarefas. Para preservar a integridade das soluções, foram desenvolvidos operadores de cruzamento específicos para representação permutacional. Neste sentido, um dos mais utilizados é o cruzamento por ordem (REEVES, 1995).

O cruzamento por ordem consiste na preservação de uma subsequência contínua de um dos pais, enquanto o restante do cromossomo é completado com os genes do outro pai, seguindo a ordem em que aparecem, ignorando as tarefas já presentes no novo cromossomo. A Figura 11 ilustra esse processo. Supondo dois indivíduos, *Pai A* e *Pai B*, cada um deles gera um novo cromossomo mantendo inalterada a porção à esquerda do ponto de corte e completando a porção à direita com os genes do outro pai, na ordem em que os elementos ausentes aparecem.

Figura 10 – Cruzamento para *Flow-shop*



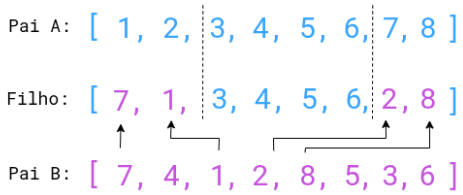
Fonte: O Autor.

A aplicação do operador de cruzamento é controlada por um parâmetro denominado probabilidade de cruzamento. Em cada geração, para cada par de pais selecionado, o cruzamento acontecerá em apenas uma parte da populção nova. Caso contrário, um dos indivíduos é simplesmente copiado para a próxima geração. Valores elevados tendem a aumentar a exploração do espaço de busca, favorecendo a recombinação de partes promissoras de soluções

distintas, enquanto valores muito baixos reduzem esse efeito e aproximam o algoritmo de uma busca baseada apenas em mutações e seleção (ALBA, 2005; HOLLAND, 1984).

Este trabalho utilizou uma variação denominada cruzamento por ordem com dois pontos, que produz um único descendente. Neste caso, selecionam-se os dois pontos de corte que delimitam um intervalo a ser copiado integralmente de um dos pais. Em seguida, as posições restantes são preenchidas com os genes do outro pai, seguindo a sua ordem relativa e ignorando elementos já presentes no filho.

Figura 11 – Cruzamento por ordem por dois pontos.

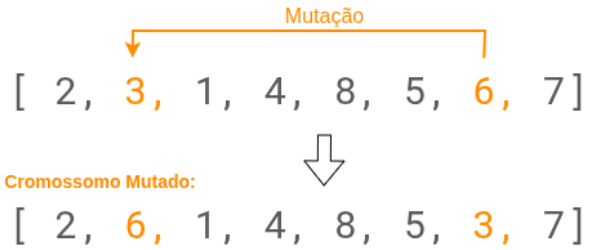


Fonte: O Autor.

3.6 MUTAÇÃO

A mutação é responsável por introduzir variações aleatórias nos cromossomos com o objetivo de manter a diversidade populacional ao longo das gerações. Sua principal função é evitar a convergência para ótimos locais, e assim possibilitando que o algoritmo continue explorando outras regiões do espaço de busca (HOLLAND, 1992; ALBA, 2005). A mutação é aplicada em apenas uma porcentagem da população, definida empiricamente na fase de experimentação. Desta forma, busca-se manter um equilíbrio entre a exploração do espaço de busca e a estabilidade evolutiva (HOLLAND, 1992). Neste trabalho, foi adotada mutação por troca, ilustrado pela Figura 12, onde dois genes são escolhidos de forma aleatória e trocados de posição.

Figura 12 – Processo de mutação



Fonte: O Autor.

3.7 SUBSTITUIÇÃO E ELITISMO

Após as operações de cruzamento e mutação, é necessário definir quais indivíduos farão parte da nova população. Esse processo é conhecido como substituição. A forma como os novos

indivíduos são incorporados influencia no equilíbrio entre a exploração de novas regiões do espaço de busca e a preservação das boas soluções (HOLLAND, 1984; GALLAGHER; SAMBRIDGE, 1994).

Existem diferentes estratégias de substituição. Em alguns casos, toda a população é descartada e substituída pelos descendentes. Embora essa abordagem possa acelerar a adaptação a novas soluções, ela também eleva o risco de perda de cromossomos de alta qualidade (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Para evitar esse risco, é comum adotar uma estratégia conhecida como elitismo, na qual um ou mais dos melhores indivíduos da população atual são preservados. O elitismo garante que a melhor solução não seja descartada, contribuindo para a estabilidade do processo evolutivo e acelerando a convergência do algoritmo (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008).

Neste trabalho adotou-se uma política de substituição parcial que é definida pelo número de descendentes criados em cada geração. A cada iteração, os novos indivíduos substituem, de forma aleatória, a mesma quantidade de membros da população, com a ressalva de que os melhores (definidos pela taxa de elitismo) não são removidos. Essa estratégia preserva diversidade, mantendo as melhores soluções.

3.8 CRITÉRIO DE PARADA

A decisão sobre quando interromper a busca é central em algoritmos genéticos, pois afeta tanto o custo computacional quanto a qualidade das soluções (HOLLAND, 1984; REEVES, 1995). Critérios simples como um número fixo de gerações são comuns, mas podem ignorar progresso efetivo, levando a interrupções prematuras ou a execuções prolongadas após a convergêcia (JAIN; BRANAVAN; BHATIA, 2001).

Este trabalho utilizou um critério amplamente adotado, baseado na convergência da solução, combinado com um limite máximo de gerações. Neste caso, quando a melhor solução não se altera ao longo de um número consecutivo de gerações, interrompe-se a execução, observando-se também um limite máximo de iterações (JAIN; BRANAVAN; BHATIA, 2001; ALBA, 2005).

3.9 ALGORITMOS GENÉTICOS PARALELOS

A execução de algoritmos genéticos em ambientes sequenciais pode se tornar custoso à medida que o problema cresce em tamanho ou complexidade (ALBA, 2005). De fato, o custo de avaliar múltiplos cromossomos por geração, aplicar operadores genéticos e repetir esse processo por diversas iterações pode resultar em tempos de execução elevados. Uma forma de reduzir esse problema é a utilização de programação paralela, visto que os algoritmos genéticos são especialmente apropriados para esse tipo de abordagem (CANTÚ-PAZ, 2000).

Existem basicamente dois modelos que são utilizados para a paralelização de algoritmos genéticos: o modelo mestre-escravo e o modelo de ilhas. Ambos os modelos apresentam características distintas em termos de desempenho, *overhead*[1] de comunicação e capacidade de preservar a diversidade genética da população (ALBA, 2005; CANTÚ-PAZ, 2000).

O modelo mestre-escravo distribui a carga computacional entre os múltiplos processadores, mantendo um processo central (mestre) responsável pelo gerenciamento do algoritmo e delegando a avaliação da função objetivo aos demais processos (escravos). Essa abordagem se trata de uma paralelização direta do algoritmo, sem alterações em sua estrutura (REEVES, 1995; CANTÚ-PAZ, 2000).

O modelo de ilhas propõe a execução simultânea de populações isoladas, chamadas de ilhas, onde cada uma delas evolui de maneira independente. Essa independência permite uma execução naturalmente paralela, reduzindo a necessidade de comunicação entre os processos. Para evitar a convergência prematura de cada ilha para ótimos locais, o modelo inclui um mecanismo de migração, no qual indivíduos selecionados de uma ilha são transferidos para as outras. Essa troca de indivíduos permite o compartilhamento de boas soluções e aumenta a diversidade global do sistema (ALBA, 2005).

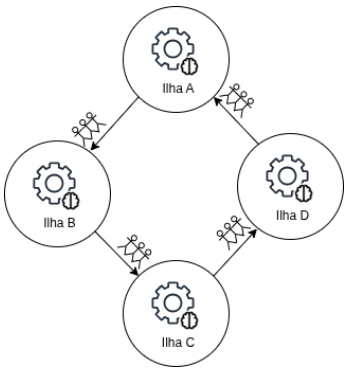
O processo de migração é definido por três elementos principais: a frequência de migração; a quantidade ou proporção de indivíduos a serem migrados; e a topologia de comunicação entre as ilhas. A frequência se refere à quantidade de gerações até que o processo de migração seja realizado. A quantidade ou proporção determina o número de indivíduos a serem migrados de cada ilha. Em geral, os indivíduos mais aptos de uma ilha são enviados a outras, substituindo os menos aptos das populações receptoras. Em algumas variações, os migrantes incluem não apenas os melhores, mas também cromossomos selecionados por outros critérios, a fim de equilibrar a disseminação de boas soluções e a manutenção da diversidade populacional (ALBA, 2005). Por fim, a topologia define como as ilhas estão conectadas, influenciando tanto o grau de diversificação global quanto o custo de comunicação entre os processos (FUKUNAGA; GONG, 2011).

A Figura 13 possui quatro ilhas (A, B, C, D), formando uma topologia do tipo anel. Neste caso, os melhores indivíduos da ilha A são migrados para a ilha B, os melhores indivíduos da ilha B são migrados para a ilha C, e assim por diante. Essa topologia favorece a preservação da diversidade genética, pois as boas soluções são propagadas de forma gradual entre as ilhas, evitando que uma única solução dominante se espalhe rapidamente por toda a população. Essa difusão lenta contribui para reduzir o risco de convergência prematura para ótimos locais. Por outro lado, a comunicação restrita a uma única vizinhança implica em uma menor velocidade de disseminação das melhores soluções, o que pode impactar negativamente o tempo necessário

[1] O termo *overhead*, em computação, refere-se ao custo adicional de tempo ou recursos que não contribui diretamente para a solução do problema em si, mas que é necessário para o controle, gerenciamento ou comunicação entre processos.

para alcançar soluções de alta qualidade global (ALBA, 2005).

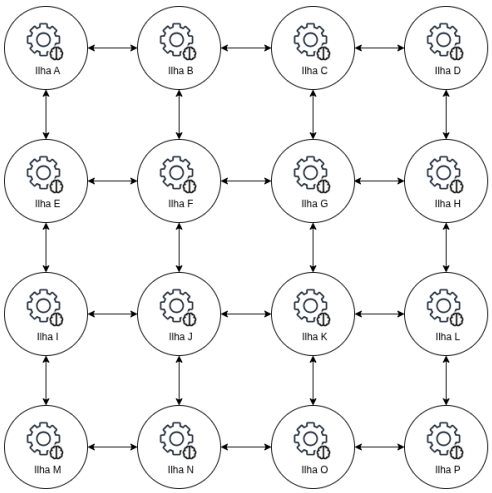
Figura 13 – Topologia em anel



Fonte: O Autor.

Outra topologia amplamente utilizada é a malha bidimensional (Figura 14). Nessas, as ilhas são organizadas em uma grade e realizam migração apenas com as ilhas adjacentes, ou seja, à esquerda, à direita, acima e abaixo. Essa configuração favorece um equilíbrio entre comunicação e isolamento parcial, permitindo que boas soluções se espalhem progressivamente por toda a população sem comprometer a exploração do espaço de busca. No entanto, esse padrão de conectividade mais denso em comparação ao anel também implica em um aumento no custo de comunicação e na complexidade de gerenciamento das trocas entre ilhas, especialmente em sistemas com grande número de processos. Portanto, embora essa topologia favoreça a diversidade e a convergência equilibrada, ela pode introduzir um *overhead* computacional significativo, que deve ser considerado na análise do desempenho geral do algoritmo (ALBA, 2005).

Figura 14 – Migração entre ilhas em malha bidimensional



Fonte: O Autor.

Neste trabalho adotou-se o modelo de ilhas, por favorecer a manutenção de uma maior diversidade e a exploração simultânea de regiões distintas do espaço de soluções. A topologia

escolhida foi a malha bidimensional, visto que ela oferece conectividade maior que o anel, o que acelera a disseminação de boas soluções, mas ainda preserva o caráter local das subpopulações De fato, o modelo em malha reduz a distância média entre ilhas e acelera a troca de informação, sem um acoplamento global como nas topologias totalmente conectadas. (ALBA, 2005; CANTÚ-PAZ, 2000; FUKUNAGA; GONG, 2011).

4 PROGRAMAÇÃO PARALELA

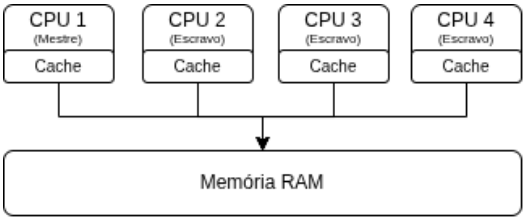
A programação paralela é uma abordagem de que baseia-se na execução simultânea de múltiplas tarefas, com o objetivo de acelerar a execução de uma aplicação. Ou seja, invés de executar as instruções de forma sequencial, como ocorre em programas tradicionais, a programação paralela permite que as partes de um problema sejam executadas simultaneamente em diferentes núcleos de processamento (GRAMA *et al.*, 2003). Essa é uma alternativa principalmente em aplicações que demandam alto poder computacional, como simulações científicas (NAVARRO; HITSCHFELD; MATEU, 2013), análise de grandes volumes de dados (CHANDRAKALA *et al.*, 2023) e resolução de problemas de otimização. De fato, ao distribuir o custo computacional entre diferentes núcleos de processamento é possível reduzir o tempo total para a execução de uma aplicação, viabilizando a resolução de problemas que seriam impraticáveis em ambientes sequenciais (LARSON *et al.*, 2009).

4.1 MEMÓRIA COMPARTILHADA E DISTRIBUÍDA

A implementação dos algoritmos genéticos pode ser realizada utilizando diferentes paradigmas de programação paralela, a depender das características do ambiente computacional disponível. Em particular, a forma como a memória é organizada determina o modelo de comunicação a ser adotado (IZZO; BISCANI; RUCIńSKI, 2010). Considerando a organização de memória, existem basicamente dois tipos de arquitetura para execução paralela: memória compartilhada e memória distribuída (RAGGER *et al.*, 1999).

Em arquiteturas com memória compartilhada, todos os processadores acessam um mesmo espaço de memória, o que facilita a comunicação e a sincronização entre os processos. A Figura 15 apresenta uma exemplo de uma arquitetura com memória compartilhada, onde todos os núcleos de processamento possuem acesso a uma mesma área de memória, facilitando assim a comunicação entre eles. Essa comunicação ocorre por meio de um barramento de memória. No entanto, esse mecanismo é eficiente apenas para pequenos volumes de acesso, pois o barramento se torna um gargalo à medida que o número de núcleos aumenta, limitando a escalabilidade da arquitetura (CHANDRAKALA *et al.*, 2023; RAGGER *et al.*, 1999). Além disso, neste tipo de arquitetura torna-se necessário a utilização de mecanismos de coerência de *cache* e de sincronização (*locks*, semáforos, barreiras, etc.) para evitar condições de corrida e garantir a ordem correta de execução das operações que possuem dependência (CHAPMAN, 2008).

Figura 15 – Memória compartilhada



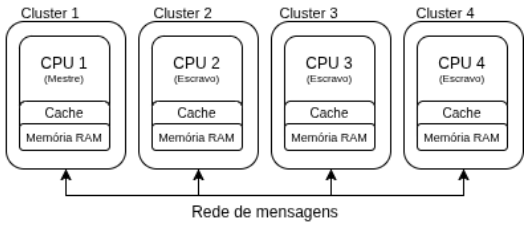
Fonte: O Autor

Entre os exemplos mais comuns de arquiteturas de memória compartilhada, destacam-se as máquinas com múltiplos processadores e com múltiplos núcleos de processamento (*multi-core*), em que vários núcleos são integrados em um único processador. Ambas as configurações são amplamente utilizadas em sistemas modernos, desde computadores pessoais até servidores corporativos (CHANDRAKALA *et al.*, 2023).

Nos ambientes com memória compartilhada a programação geralmente é realizada através da utilização de múltiplas *threads*. De fato, nesses ambientes é comum a utilização de bibliotecas de *threads* como o OpenMP (*Open specifications for Multi Processing*) que fornece uma interface de alto nível para paralelismo baseado em *threads* (CHAPMAN, 2008), ou ainda o uso de bibliotecas como a Pthreads (POSIX *Threads*), que oferecem um controle mais flexível para a criação e gerenciamento de múltiplas *threads* (BUTENHOF, 1997).

Por outro lado, em arquiteturas de memória distribuída (Figura 16), cada processador possui sua própria memória local, e a comunicação entre processos ocorre por meio da troca de mensagens sobre uma rede de computadores. Essa abordagem impõe um *overhead* de comunicação superior ao do modelo de memória compartilhada, em razão da maior latência da rede (GRAMA *et al.*, 2003).

Figura 16 – Memória distribuída



Fonte: O Autor

Esse modelo é característico em *clusters* de computadores, onde os nós são independentes e interligados por uma rede de computadores. A ausência de memória compartilhada entre os nós exige o uso de uma biblioteca de troca de mensagens para a comunicação entre os processos (RAGGER *et al.*, 1999). Para a comunicação entre processos, tradicionalmente são utilizadas as bibliotecas MPI (*Message Passing Interface*) (GROPP; LUSK; SKJELLUM, 1999) e PVM (*Parallel Virtual Machine*) (GEIST *et al.*, 1995).

Neste trabalho, optou-se por utilizar uma arquitetura de memória distribuída, mais especificamente um *cluster* de computadores. A escolha por *clusters* justifica-se principalmente por sua maior escalabilidade e custo-benefício em relação a ambientes com memória compartilhada (RAGGER *et al.*, 1999). De fato, arquiteturas com memória compartilhada são mais difíceis de serem expandidas, devido à saturação do barramento de memória, aumento da contenção devido ao acesso simultâneo à mesma memória, aumento do consumo energético, aumento do calor gerado, entre outros fatores (GRAMA *et al.*, 2003). Por outro lado, os *clusters* de computadores permitem aumentar a capacidade computacional pela adição de novos nós à rede, geralmente com *hardware* comum e de menor custo. Essa flexibilidade torna os *clusters* de computadores uma alternativa para a execução de algoritmos paralelos de maior porte, oferecendo uma boa relação entre desempenho e o investimento necessário (SKORPIL; OUJEZSKY, 2022; DONGARRA; HEMPEL; WALKER, 1993).

Para a comunicação entre processos, foi utilizada a biblioteca de troca de mensagens MPI, um padrão amplamente adotado em aplicações de alto desempenho. A escolha pelo uso do MPI se justifica por sua ampla aceitação na comunidade de computação de alto desempenho, sua portabilidade entre diferentes plataformas e sua eficiência em *clusters* de computadores (SKORPIL; OUJEZSKý; TULEJA, 2020). Outra alternativa seria o uso do PVM (*Parallel Virtual Machine*), amplamente utilizado antes da padronização do MPI, porém o uso dessa foi desconsiderado devido à sua baixa utilização atualmente.

4.2 CLUSTERS DE COMPUTADORES

Os *clusters* de computadores surgiram como uma alternativa de baixo custo para atender às crescentes demandas computacionais de aplicações científicas e industriais. O desenvolvimento dos *clusters* ganhou força a partir da década de 1990, impulsionado pela crescente disponibilidade de computadores com preços acessíveis e pela evolução das redes de locais de comunicação (BUYYA, 1999).

A arquitetura dos *clusters* de computadores foi popularizada com o projeto Beowulf, desenvolvido em 1994 na *National Aeronautics and Space Administration* (NASA). O projeto Beowulf utilizou computadores pessoais convencionais conectados por uma rede *Ethernet* para formar um sistema de alto desempenho, mostrando que era possível alcançar um alto poder computacional sem o uso de supercomputadores proprietários e de alto custo (BECKER *et al.*, 1995). Assim, a arquitetura Beowulf se consolidou como uma referência para o desenvolvimento dos *clusters* de computadores. A Figura 17 apresenta uma imagem do *cluster* desenvolvido no projeto Beowulf da NASA.

Figura 17 – *Cluster* com arquitetura Beowulf

Fonte: Adaptado de (FOUNDATION, 2025)

A ideia central de um *cluster* é agrupar diversos computadores independentes para atuarem de forma cooperativa para a execução de tarefas paralelas ou distribuídas. Neste caso, cada nó em um *cluster* corresponde a um computador completo, operando com sua própria memória e sistema operacional. A comunicação entre os nós é realizada por meio de uma rede, geralmente utilizando protocolos de comunicação padronizados, como por exemplo, o TCP/IP (*Transmission Control Protocol/Internet Protocol*). Neste caso, a troca de informações entre os processos é realizada através de mecanismos explícitos de comunicação, como o envio de mensagens através de bibliotecas específicas, como por exemplo, o MPI (BUYYA, 1999).

Os *clusters* de computadores são altamente escaláveis, permitindo o aumento da capacidade computacional por meio da simples adição de novos nós à rede. Essa escalabilidade contrasta com as limitações físicas arquiteturais encontradas em sistemas com memória compartilhada, que geralmente enfrentam gargalos de barramento, contenção por acesso simultâneo à memória, e restrições energéticas e térmicas (GRAMA *et al.*, 2003). Além disso, *clusters* oferecem uma ótima relação custo-benefício, pois podem ser compostos por máquinas comuns (por exemplo, computadores pessoais ou servidores simples), conectadas por uma rede de computadores com baixo custo. Isso torna essa arquitetura atrativa para instituições de pesquisa e ambientes acadêmicos, que demandam de alto desempenho sem investimentos elevados (NGXANDE; MOOROSI, 2014).

A principal desvantagem dos *clusters* de computadores reside no custo adicional de comunicação. De fato, a troca de informações entre processos exige operações de envio e recebimento por meio da rede, o que acarreta em uma maior latência em comparação com acessos a uma memória compartilhada. Esse fator pode impactar negativamente no desempenho de

aplicações que exigem comunicação frequente (BUYYA, 1999; RAGGER *et al.*, 1999). Assim, os *clusters* de computadores não são adequados para tarefas de baixa granularidade, ou seja, tarefas muito pequenas que exigem pouca computação por unidade de trabalho. Nesses casos, o custo da comunicação pode superar o ganho obtido com a paralelização (MARTIN *et al.*, 1997).

Apesar de oferecerem uma boa relação custo-benefício, os *clusters* de computadores ainda demandam uma infraestrutura física adequada, com rede dedicada, fornecimento elétrico estável e refrigeração apropriada. Esses requisitos podem representar um desafio logístico e de custos, embora o custo total ainda seja significativamente inferior ao de supercomputadores proprietários e de alto desempenho. Dessa forma, os *clusters* mantêm-se como uma solução viável e economicamente atrativa para a execução de aplicações científicas e acadêmicas que exigem grande capacidade de processamento (CHANDRAKALA *et al.*, 2023; MARTIN *et al.*, 1997).

4.3 MPI - *MESSAGE PASSING INTERFACE*

Para que os nós de um *cluster* possam cooperar é necessário um mecanismo de comunicação entre os processos. Nesse contexto, destaca-se o MPI, uma biblioteca padronizada e amplamente utilizada que provê os recursos necessários para troca de mensagens e coordenação entre os processos que compõem uma aplicação paralela (SKORPIL; OUJEZSKý; TULEJA, 2020; ZOUNMEVO *et al.*, 2018; GRAMA *et al.*, 2003). A popularidade do MPI se deve à combinação de desempenho, escalabilidade e flexibilidade. Além disso, o MPI é altamente portável e compatível com diferentes plataformas, características que contribuíram para sua consolidação como o principal padrão de comunicação em sistemas paralelos de alto desempenho.

O padrão MPI define um conjunto de funções que permite a execução paralela cooperativa entre múltiplos processos, mesmo que estejam distribuídos em diferentes computadores de uma rede. A forma mais comum de utilização do MPI é seguindo o paradigma conhecido como *Single Program, Multiple Data* (SPMD), no qual todos os processos executam o mesmo programa, mas operam sobre diferentes dados. Neste caso, cada processo possui um identificador único e o comportamento de cada processo pode ser diferenciado com base nesse identificador, permitindo especificar as instruções que cada um deve executar (GRAMA *et al.*, 2003; GROPP; LUSK; SKJELLUM, 1999).

Entre as principais funcionalidades do MPI estão: mecanismos de envio e recebimento de mensagens ponto a ponto; operações coletivas de comunicação; e rotinas de sincronização entre processos (DONGARRA; HEMPEL; WALKER, 1993; GROPP; LUSK; SKJELLUM, 1999; ZHOU; GRACIA; SCHNEIDER, 2020).

A comunicação ponto a ponto permite o envio e recebimento de mensagens entre dois processos específicos. Essa modalidade é fundamental para a troca de mensagens entre pares de processos, oferecendo suporte para a troca de informações de forma bloqueante e não bloqueante. Esse tipo de comunicação é apropriada para algoritmos que executam trocas diretas de

informações entre os processos, como no modelo mestre-escravo, em que um processo mestre distribui tarefas para escravos e coleta os resultados individualmente (GROPP; LUSK; SKJELLUM, 1999).

A comunicação coletiva envolve múltiplos processos simultaneamente e é essencial em operações que requerem cooperação entre todos os processos da aplicação, como disseminação de dados, agregação de resultados ou sincronização de estados. Um exemplo é a operação de *broadcasting*, que transmite uma mensagem de um processo para todos os demais. Outra operação coletiva amplamente empregada é a redução, a qual realiza operações, como soma ou máximo, sobre dados provenientes de múltiplos processos, consolidando o resultado em um único processo (ZHOU; GRACIA; SCHNEIDER, 2020).

Por fim, a sincronização entre processos garante a coordenação do fluxo de execução em programas paralelos. O MPI oferece primitivas como barreiras, que bloqueiam todos os processos participantes até que todos tenham alcançado um ponto específico do código. Essa funcionalidade é essencial para manter a consistência do estado global e evitar condições de corrida, especialmente em seções do programa que dependem da conclusão simultânea de múltiplas tarefas para prosseguir (GROPP; LUSK; SKJELLUM, 1999).

5 IMPLEMENTAÇÃO DESENVOLVIDA

Este capítulo apresenta a implementação do algoritmo genético para o *Flow-Shop* permutacional. Inicialmente, é descrita a versão sequencial, abordando as estruturas de dados utilizadas e a implementação dos operadores genéticos. Em seguida, apresenta-se a implementação paralela baseada no modelo de ilhas e desenvolvida com o uso da biblioteca MPI.

5.1 ALGORITMO GENÉTICO SEQUENCIAL

A implementação sequencial foi desenvolvida utilizando a linguagem de programação C padrão ANSI. Esse tem início pela leitura dos dados da instância do problema de escalonamento do tipo *Flow-Shop* permutacional. Os dados devem estar armazenados em um arquivo texto no formato estabelecido por Taillard (1993). Neste, a primeira linha contém a quantidade de tarefas e de máquinas. Em seguida, em cada linha (uma por máquina) tem-se os tempos de cada tarefa. No Listagem 1 é possível ver um exemplo onde na primeira linha tem-se uma instância com 20 tarefas e 5 máquinas. Em seguida, cada linha apresenta o tempo de processamento das 20 tarefas em cada máquina. Na implementação desenvolvida, os dados lidos do arquivo são armazenados em uma matriz bidimensional, onde cada linha representa uma máquina e as colunas as tarefas.

Listagem 1 – Formato Taillard (1993)

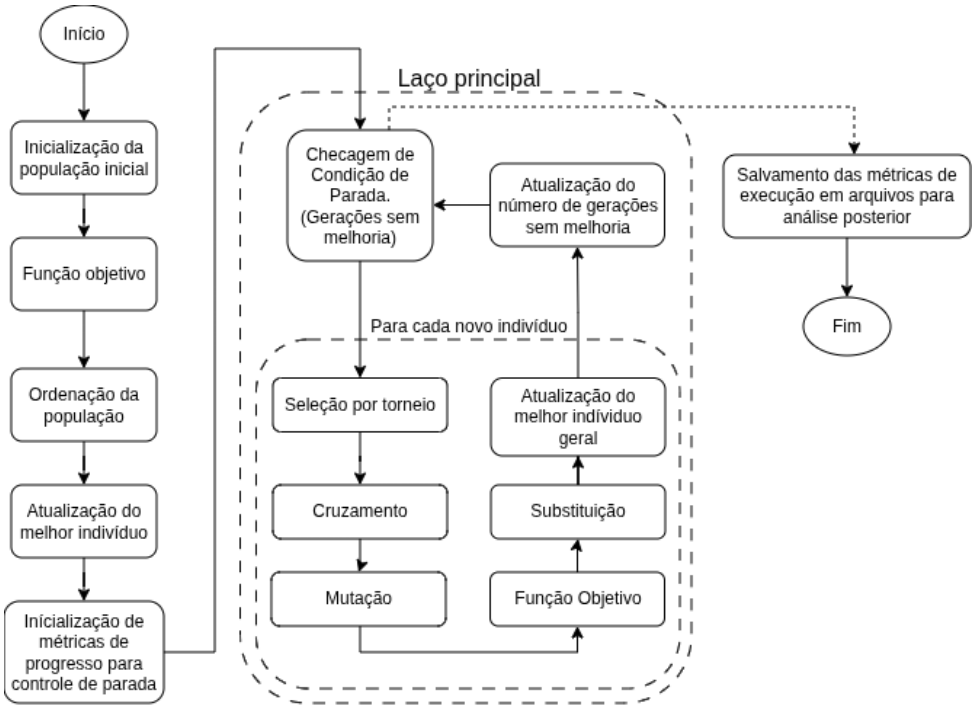
```
1 20 5
2 54 83 15 71 77 36 53 38 27 87 76 91 14 29 12 77 32 87 68 94
3 79  3 11 99 56 70 99 60  5 56  3 61 73 75 47 14 21 86  5 77
4 16 89 49 15 89 45 60 23 57 64  7  1 63 41 63 47 26 75 77 40
5 66 58 31 68 78 91 13 59 49 85 85  9 39 41 56 40 54 77 51 31
6 58 56 20 85 53 35 53 41 69 13 86 72  8 49 47 87 58 18 68 28
```

Após a etapa de leitura dos dados, inicia-se a execução do algoritmo genético, o qual segue a estrutura tradicional (Figura 18). Primeiramente, realiza-se a inicialização da população, que corresponde ao conjunto de soluções viáveis para o problema. Também são inicializadas as estruturas utilizadas para o controle de parada. Em seguida, cada indivíduo é avaliado conforme a função objetivo, que calcula o tempo total de execução da solução. Após essa etapa, a população é ordenada usando o algoritmo *quicksort*, e o indivíduo com menor tempo total de execução é armazenado, sendo tratado neste trabalho como o melhor indivíduo.

No laço principal, a cada geração é verificada a condição de parada, baseada no número de gerações consecutivas sem melhoria no melhor indivíduo armazenado. Caso o critério de convergência não seja atendido, o contador de gerações sem melhoria é atualizado. A formação da próxima geração segue as seguintes etapas: seleção por torneio para escolha dos pais; cruzamento para recombinar partes das soluções; mutação para introdução de variações; avaliação

dos descendentes com base na função objetivo; e substituição, na qual novos indivíduos são inseridos na população não protegida por elitismo. Ao final desse ciclo, o melhor indivíduo é atualizado sempre que uma solução com tempo total de execução inferior for encontrada.

Figura 18 – Algoritmo genético sequencial



Fonte: O Autor.

Na implementação desenvolvida, cada solução candidata é representada por um vetor de inteiros que descreve uma permutação de tarefas. A posição correspondente do vetor indica a ordem de processamento da tarefa. A população é formada por uma coleção de vetores, conforme apresentado na Figura 19.

Figura 19 – Armazenamento da população

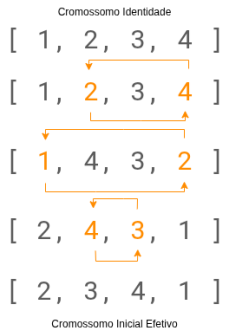
```
[
  [ 2, 1, 4, 3 ],
  [ 4, 1, 3, 2 ],
  [ 2, 4, 3, 1 ],
  [ 2, 3, 4, 1 ],
  [ 3, 2, 4, 1 ],
  [ 3, 4, 2, 1 ],
]
```

Fonte: O Autor.

A população inicial é gerada de forma aleatória. Para isso, parte-se de uma lista ordenada de tarefas, chamada de cromossomo identidade, e aplica-se um algoritmo de embaralhamento, que troca os elementos de posição. A quantidade de trocas realizadas é proporcional

ao número de tarefas presentes no cromossomo. Esse procedimento garante que não existam repetições de tarefas dentro de cada indivíduo. A Figura 20 ilustra o processo de embaralhamento para criação de um cromossomo. Esse processo pode gerar indivíduos com a mesma permutação, porém isso não compromete o funcionamento do método, pois a diversidade tende a aumentar nas gerações seguintes com a aplicação das operações de seleção, cruzamento e mutação.

Figura 20 – Criação dos cromossomos via embaralhamento



Fonte: O Autor.

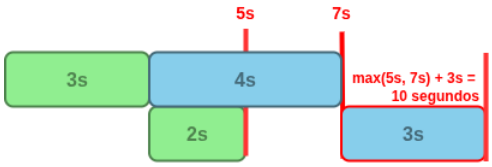
A função objetivo retorna o tempo total de execução das tarefas de uma solução representada pelo cromossomo. Em *Flow-Shop*, cada tarefa passa por todas as máquinas na mesma ordem. A Figura 22 ilustra o cálculo da função objetivo para a permutação cuja ordem de execução é: *Tarefa 2*, *Tarefa 1*, *Tarefa 4* e *Tarefa 3*. Nesse exemplo, o tempo total de execução corresponde ao instante em que a última tarefa da sequência é concluída na última máquina. Dessa forma, o valor da função objetivo é definido pelo término da *Tarefa 3* na *Máquina 4*, resultando em um tempo total de 18 segundos.

O cálculo matemático da função objetivo segue uma lógica simples. Para iniciar uma subtarefa em uma máquina, duas condições precisam estar verdadeiras ao mesmo tempo: a própria tarefa já deve ter terminado na máquina anterior e a máquina atual deve estar livre. O início ocorre no instante mais tarde entre esses dois momentos, e o término é esse início somado à duração da subtarefa. Em outras palavras, a subtarefa espera o que for necessário, seja a chegada da tarefa vinda da máquina anterior, seja a liberação da máquina atual.

A Figura 21 mostra esse princípio em um caso simples. A subtarefa azul dura três segundos. A tarefa chega da máquina anterior aos cinco segundos, porém a máquina atual só fica livre aos sete segundos. Como é preciso respeitar as duas condições, a execução começa aos sete e termina aos dez segundos.

Aplicando o mesmo procedimento para todas as barras, construímos o gráfico completo da Figura 22, correspondente à permutação *Tarefa 2, Tarefa 1, Tarefa 4, Tarefa 3*. Observe três momentos que deixam a lógica clara. Na Máquina 1, a Tarefa 1 dura três segundos e começa aos dois segundos, quando a máquina fica disponível, terminando aos cinco. Na Máquina 2, a Tarefa 3 dura três segundos, chega dessa mesma tarefa da máquina anterior aos doze segundos

Figura 21 – Tempo Final de uma Subtarefa

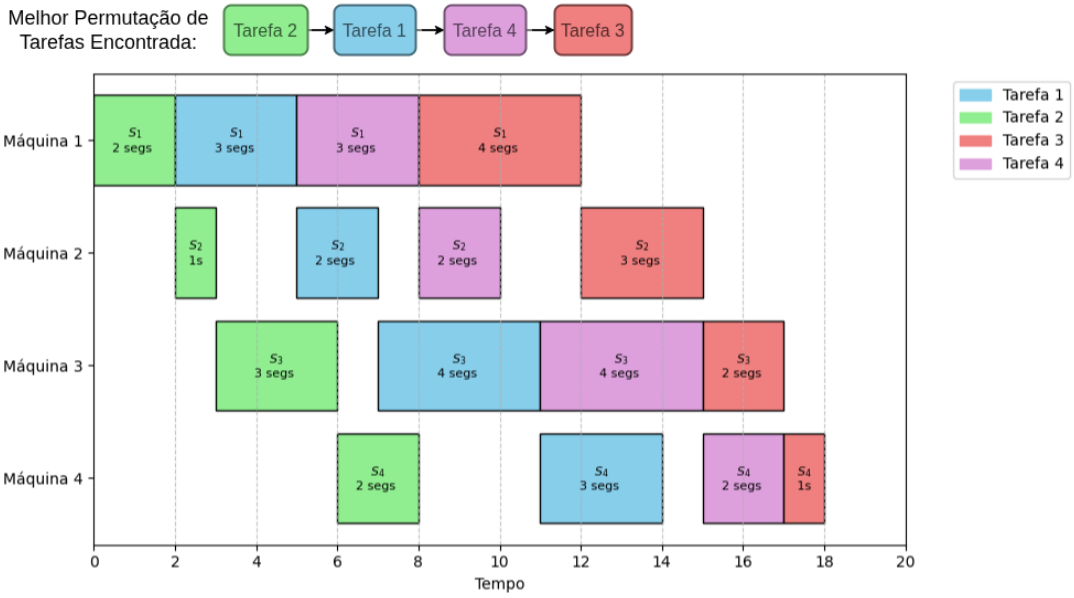


Fonte: O Autor.

e encontra a máquina livre no mesmo instante, por isso termina aos quinze. Na Máquina 4, a última subtarefa da Tarefa 3 dura um segundo e apenas pode iniciar aos dezessete, instante em que a máquina está livre e a tarefa chega de M3. Ao finalizar essa barra, o relógio marca dezoito segundos.

O valor da função objetivo é exatamente esse último término observado na última máquina. Para a sequência ilustrada, o tempo total de execução é de dezoito segundos, que coincide com o final da última barra do gráfico.

Figura 22 – Avaliação da função objetivo.



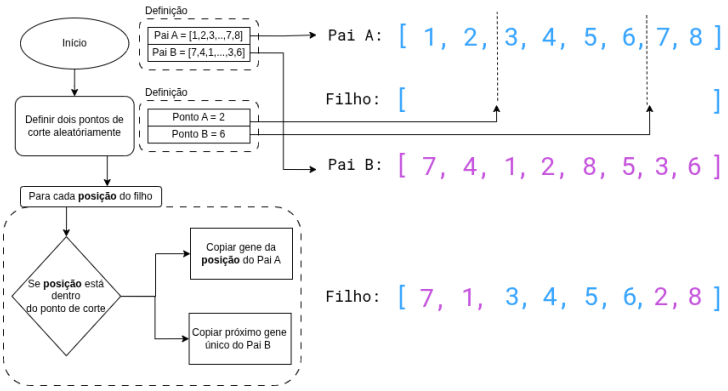
Fonte: O Autor.

A seleção dos pais é realizada pelo método de torneio binário. Em cada torneio, dois indivíduos são escolhidos aleatoriamente na população e compara-se o valor da função objetivo de cada um. O indivíduo com menor tempo total de execução é selecionado para compor o conjunto de pais. Esse processo é repetido até que o número necessário de pais seja obtido para o cruzamento. O método de torneio foi adotado por sua simplicidade e eficiência, além de manter um bom equilíbrio entre pressão seletiva e diversidade populacional.

O cruzamento é realizado utilizando o cruzamento por ordem, na sua variação de dois pontos de corte (REEVES, 1995). Conforme pode ser observado na Figura 23, primeiramente,

selecionam-se dois pontos aleatórios no primeiro pai e copia-se o segmento compreendido entre eles para o descendente. Em seguida, as posições restantes são preenchidas com os genes do segundo pai, mantendo-se a ordem original e evitando duplicações.

Figura 23 – Função de cruzamento



Fonte: O Autor.

Após a geração de cada descendente, procede-se à substituição dos indivíduos, mantendo constante o tamanho da população. Neste caso, adotou-se um elitismo, onde uma fração pré-determinada por parâmetros dos melhores indivíduos é preservada integralmente. Essa fração é um valor predefinido no início da execução do algoritmo genético. Em seguida, os novos piores indivíduos são substituídos até completar a população.

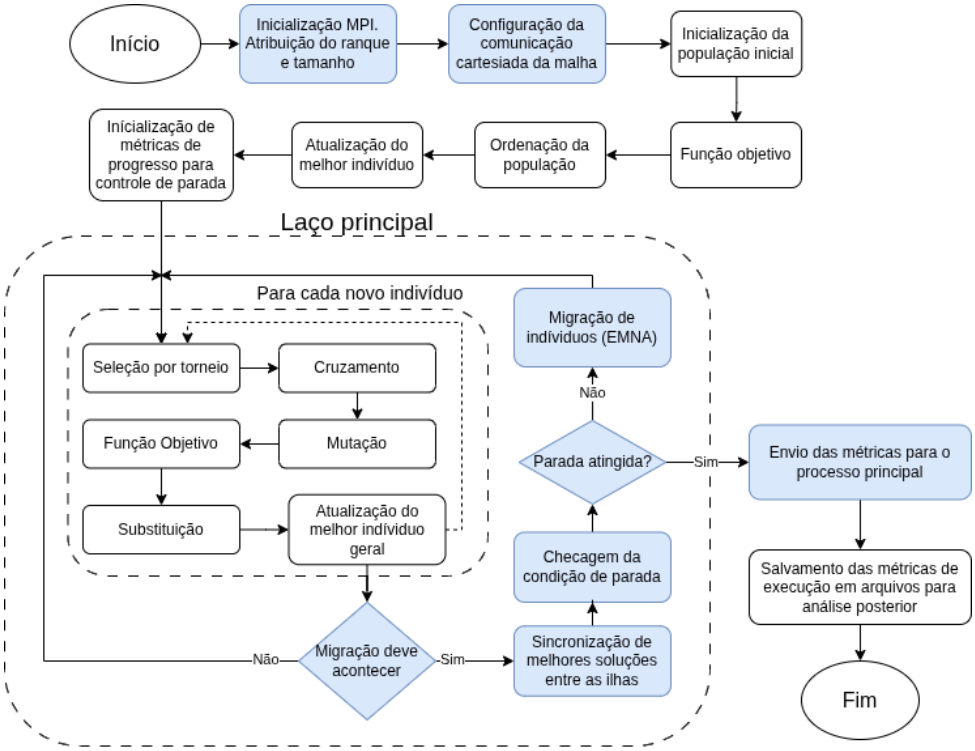
A mutação foi implementada por meio do operador de troca, no qual duas posições aleatórias do cromossomo têm seus genes trocados entre si. A aplicação desse operador é controlada por uma taxa de mutação pré-definida. Essa taxa determina a proporção de indivíduos da população que sofrerão mutação a cada geração, sendo ajustada experimentalmente.

A condição de parada do algoritmo é atingida quando a melhor solução não se modifica durante um número consecutivo de gerações predefinido, ou pelo atingimento de um limite máximo de gerações. O controle desse critério é realizado por meio de um contador, que é incrementado a cada iteração em que a melhor solução permanece inalterada. Quando uma nova solução de melhor desempenho é encontrada, o contador é reiniciado.

5.2 IMPLEMENTAÇÃO DO MODELO DE ILHAS

A paralelização foi realizada utilizando a biblioteca *Open Message Passing Interface* (OpenMPI) na sua versão 4.1.2, que implementa o padrão MPI versão 3.1. No modelo em ilhas, cada processo executa uma instância independente do algoritmo genético, que corresponde a uma ilha. A Figura 24 apresenta a estrutura básica do modelo implementado, onde o fluxo básico do algoritmo permanece idêntico ao sequencial, e as extensões paralelas são destacadas em azul

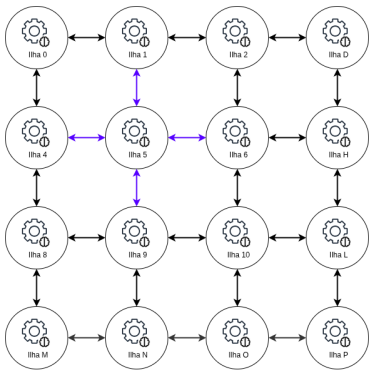
Figura 24 – Implementação do Modelo em Ilhas



Fonte: O Autor.

A inicialização do MPI corresponde à obtenção dos identificadores dos processos (função `MPI_Comm_rank`) e do número total de processos (função `MPI_Comm_size`). Após, tem-se a configuração de uma topologia virtual, que corresponde a uma topologia cartesiana bidimensional criada através da função `MPI_Cart_create`. Essa estrutura define uma topologia lógica em grade, na qual cada processo é associado a uma coordenada e seus vizinhos diretos são obtidos com chamadas a `MPI_Cart_shift`. Por exemplo, na Figura 25, considerando uma configuração de 16 processos organizados em uma grade 4×4, o processo de identificador 5 estaria localizado na posição $(1, 1)$ da malha, tendo como vizinhos diretos os processos de identificadores 1 (acima), 9 (abaixo), 4 (à esquerda) e 6 (à direita). Essa vizinhança é automaticamente identificada pela operação `MPI_Cart_shift`, que retorna os identificadores dos processos vizinhos, permitindo que cada processo estabeleça comunicação apenas com seus vizinhos definidos pela topologia.

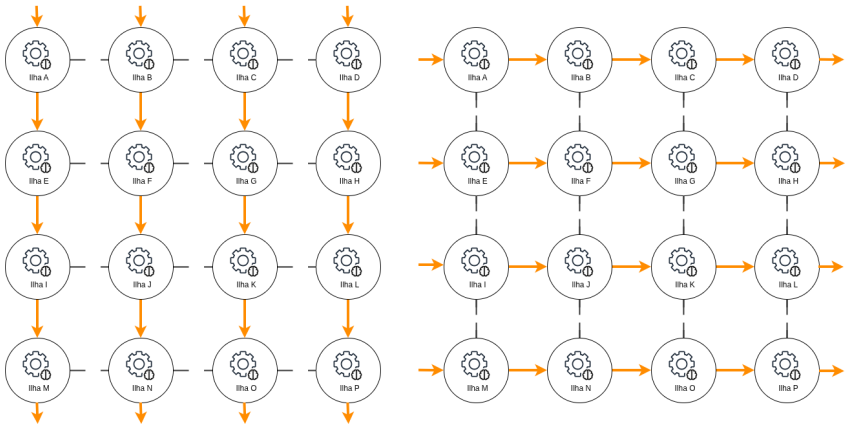
Figura 25 – Topologia Virtual



Fonte: O Autor.

A migração ocorre de forma periódica em um número parametrizado de gerações definido previamente. Durante esse período, cada subpopulação evolui de maneira independente, permitindo a exploração local do espaço de busca antes da troca de informações genéticas. Após o alcance desse intervalo, inicia-se o evento de migração, que é conduzido em duas fases distintas e síncronas: primeiro envolvendo os processos vizinhos no eixo vertical e, em seguida, no eixo horizontal (Figura 26). Essa divisão em duas etapas garante que, em cada fase, o tráfego de mensagens ocorra apenas entre pares exclusivos de processos, evitando dependências circulares e possíveis condições de bloqueio (*deadlocks*[4]) durante a comunicação (CANTÚ-PAZ, 2000; ALBA, 2005). Cada troca é realizada por meio da função `MPI_Sendrecv`, que efetua o envio e o recebimento de dados de forma simultânea entre os pares definidos pela topologia. Entre as duas etapas, são aplicadas barreiras de sincronização (`MPI_Barrier`) para garantir que todas as trocas tenham sido concluídas antes do início da próxima fase.

Figura 26 – Migração: fase vertical e fase horizontal



Fonte: O Autor.

[4] O termo *deadlock* refere-se a uma condição de bloqueio mútuo em sistemas concorrentes, na qual dois ou mais processos ficam indefinidamente esperando por recursos ou mensagens uns dos outros, impedindo o prosseguimento da execução. Em comunicações MPI, isso ocorre quando processos utilizam operações de envio e recebimento bloqueantes sem uma ordem de sincronização adequada (GRAMA *et al.*, 2003).

O critério de parada considera a estagnação em cada ilha ou na população global. Para monitorar a estagnação global, é obtido o valor do melhor indivíduo de todas as ilhas utilizando a operação `MPI_Allreduce`, com o parâmetro `MPI_MIN`. A execução é encerrada quando o melhor valor global permanece inalterado por um número consecutivo de gerações. Além disso, cada processo verifica se sua população entrou em estagnação local (plateau) e comunica esse estado aos demais. A execução é encerrada quando o número de ilhas em plateau atinge o quórum[5] definido de 75% das ilhas.

Para a seleção dos indivíduos migrantes, foi utilizada uma política híbrida denominada Elite, Novidade, Mutado, Aleatório (ENMA) (ALBA, 2005; EIBEN; SMITH, 2015). Essa política é composta por quatro grupos de indivíduos. O primeiro grupo, denominado elite (E), consiste nos melhores indivíduos da população local. O segundo grupo, denominado novidade (N), é formado por indivíduos estruturalmente distintos da elite local. Para a avaliação dos indivíduos foi utilizado o coeficiente de correlação de postos de Spearman (ZAEFFERER, 2014)[1]. Essa métrica garante que os indivíduos selecionados para migração não sejam redundantes em relação à elite, mantendo a diversidade entre as ilhas. O terceiro grupo, elite mutada (M), representa versões perturbadas da elite, produzidas por mutações leves. Por fim, o grupo aleatório (A) é composto por indivíduos selecionados aleatoriamente.

Após a recepção dos migrantes, a substituição dos indivíduos locais é conduzida por um mecanismo de aglomeração, que prioriza a substituição de indivíduos semelhantes aos imigrantes, identificados através do coeficiente de correlação de postos de Spearman (REEVES, 1995). Esse processo é complementado por uma política de restrição temporal de migração, que impede que os indivíduos recém recebidos sejam imediatamente reenviados em novas trocas.

[5] O termo quórum refere-se à fração mínima de processos que precisam atender a uma determinada condição simultaneamente. Neste caso, representa a proporção de ilhas em estagnação local necessária para disparar a parada coletiva.

[1] Mede o desvio posicional entre duas permutações, somando as diferenças absolutas entre as posições correspondentes. Por exemplo, considere dois cromossomos representados pelas permutações $A = [1, 2, 3, 4, 5]$ e $B = [2, 1, 3, 5, 4]$. A soma das diferenças de posição é $|1 - 2| + |2 - 1| + |3 - 3| + |4 - 5| + |5 - 4| = 4$. Esse valor indica a distância estrutural entre as duas soluções — quanto maior o valor, mais distintas são as permutações.

6 TESTES E ANÁLISE DE RESULTADOS

Este capítulo apresenta uma análise dos principais resultados obtidos neste trabalho. Inicialmente são apresentados os resultados da versão sequencial do algoritmo genético. Em seguida, são apresentados os resultados do modelo em ilhas, considerando diferentes arranjos de migração. Por fim, é apresentada uma comparação entre as versões sequencial e o modelo em ilhas.

6.1 BASE DE TESTES

Para os testes, foi utilizada a instância *ta111* de *Flow-Shop* permutacional composta por 500 tarefas e 20 máquinas, gerada a partir da semente 1368624604 proposta por Taillard (1993). Essa coleção de instâncias é amplamente utilizada como referência na literatura por representar casos de alta complexidade computacional. O autor reporta, para essa instância, um melhor tempo total de processamento das tarefas de 26.699 unidades, valor denominado como limite superior (*Upper Bound* (UB)), obtido por meio de um algoritmo metaheurístico de busca tabu.

6.2 RESULTADOS DA IMPLEMENTAÇÃO SEQUENCIAL

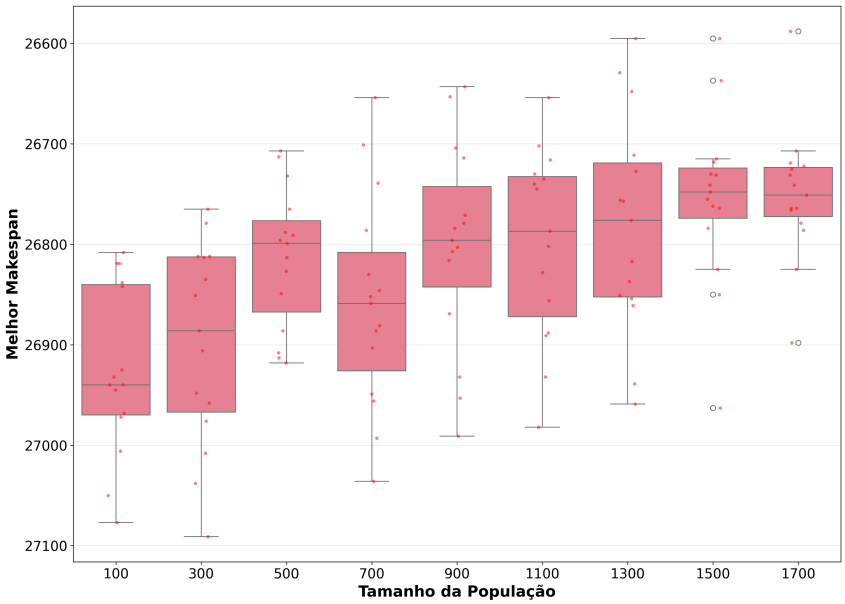
O desempenho de um algoritmo genético é fortemente influenciado por diversos parâmetros, entre os quais se destacam: o tamanho da população, o critério de convergência, a quantidade de indivíduos novos introduzidos a cada geração, as probabilidades de cruzamento e mutação, o tamanho do torneio e a taxa de elitismo. Cada um desses fatores impacta diretamente na qualidade média das soluções e o tempo de convergência (ALBA, 2005; REEVES, 1995; HOLLAND, 1992). Os efeitos desses parâmetros são apresentados nesta seção. Para tanto, cada configuração foi executada 15 vezes, sendo reportada a média do melhor tempo total de execução das tarefas, o desvio padrão entre replicações, o melhor valor observado e o número médio de gerações até o término. Após o termino de cada execução, o melhor valor definido do parâmetro foi utilizado nos testes subsequentes.

6.2.1 Tamanho da População

A Figura 27 apresenta o impacto do tamanho da população sobre o melhor tempo total das tarefas. Observa-se uma melhora mais significativa ao aumentar o tamanho da população de 100 para 500 indivíduos, com uma redução média de aproximadamente 110 unidades de tempo. A partir desse ponto, os ganhos tornam-se menores. Entre populações de 900 e 1.700 indivíduos, a diferença média é inferior a 60 unidades, indicando que o aumento do custo computacional

não se justifica pelos ganhos obtidos. Assim, optou-se pela utilização de uma população de 900 indivíduos para os demais testes.

Figura 27 – Influência do Tamanho da População



Fonte: O Autor.

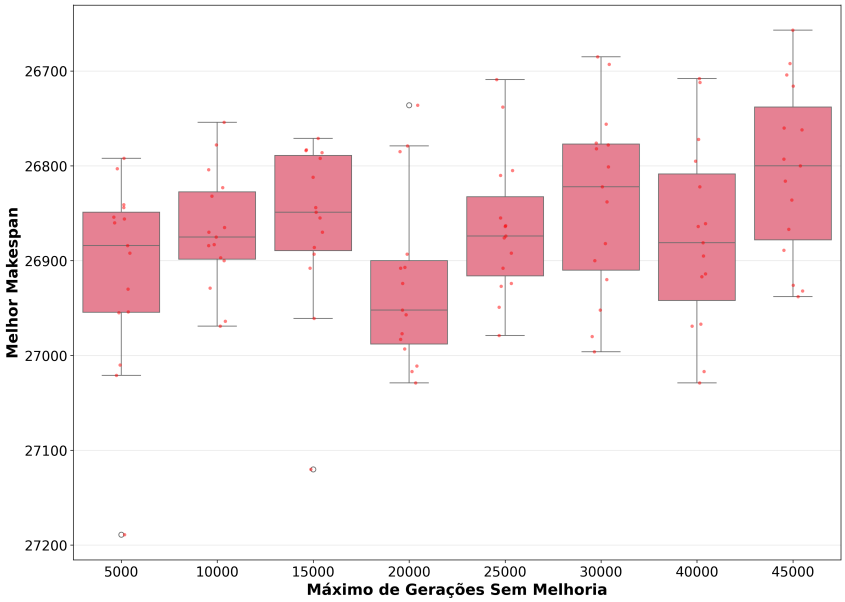
Esse comportamento é também é observado em estudos prévios sobre algoritmos genéticos aplicados ao problema de *Flow-Shop*, que também relatam ganhos expressivos apenas nas faixas iniciais de aumento da população e saturação a partir de valores intermediários (REEVES, 1995; ARMENTANO; RONCONI, 1999; RUIZ; MAROTO, 2005). Além disso, essa configuração atingiu um tempo de execução médio (valor da função objetivo) de $26801,0$. Observa-se que o valor obtido é ligeiramente inferior ao UB de 26.699 unidades. Contudo, em $13,3\%$ das réplicas, esse limite foi superado.

6.2.2 Critério de Convergência

Foram realizados testes considerando como critério de parada o número máximo de iterações consecutivas sem melhoria antes de encerrar a execução. Valores baixos interrompem a busca precocemente, enquanto valores altos prolongam a exploração, porém com maior custo de tempo. A Figura 28 mostra que o aumento do limiar de estagnação tende a melhorar a qualidade média das soluções, mas com ganhos cada vez menores. Com um limite de 30.000 gerações, obteve-se um valor médio da função objetivo de $26.837,40$. Ao aumentar o limite para 45.000 gerações, o valor médio passou para $26.805,87$, o que representa uma melhoria de apenas $0,12\%$ na qualidade da solução produzida pelo algoritmo genético. Em contrapartida, o tempo de execução das simulações aumentou cerca de 35%. Dessa forma, optou-se pela utilização de um limite de 30.000 gerações, por representar um equilíbrio entre qualidade da solução

e custo computacional. Nestas configurações, 13.3% das replicações ultrapassaram o limite de referência UB de 26.699 unidades.

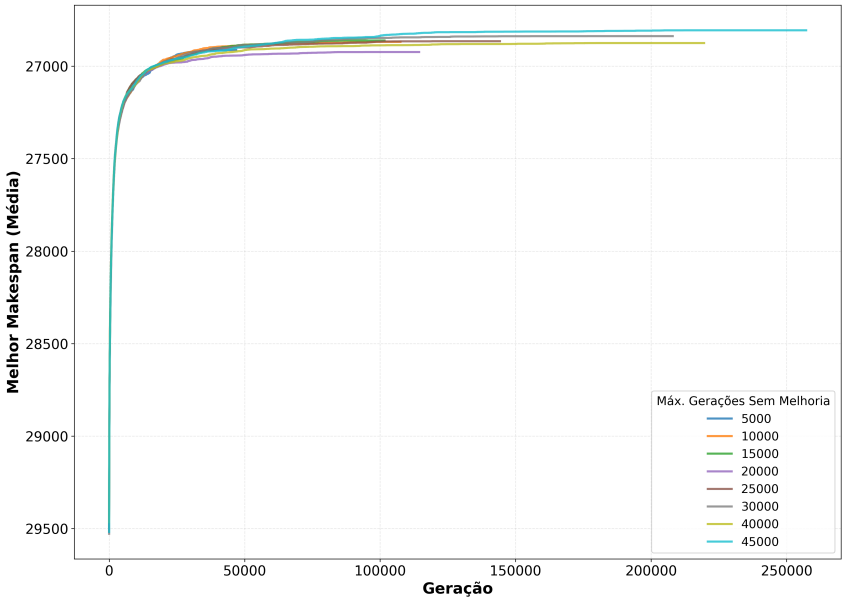
Figura 28 – Avaliação do Critério de Convergência



Fonte: O Autor.

A Figura 29 apresenta a convergência ao longo das gerações para diferentes limiares de estagnação. Observa-se, que as curvas apresentam comportamento semelhante, convergindo rapidamente e estabilizando por volta de 50.000 gerações. Limiares mais altos apenas aumentam o tempo de execução, resultando em ganhos pequenos.

Figura 29 – Melhor indivíduo em Função do Número de Gerações.



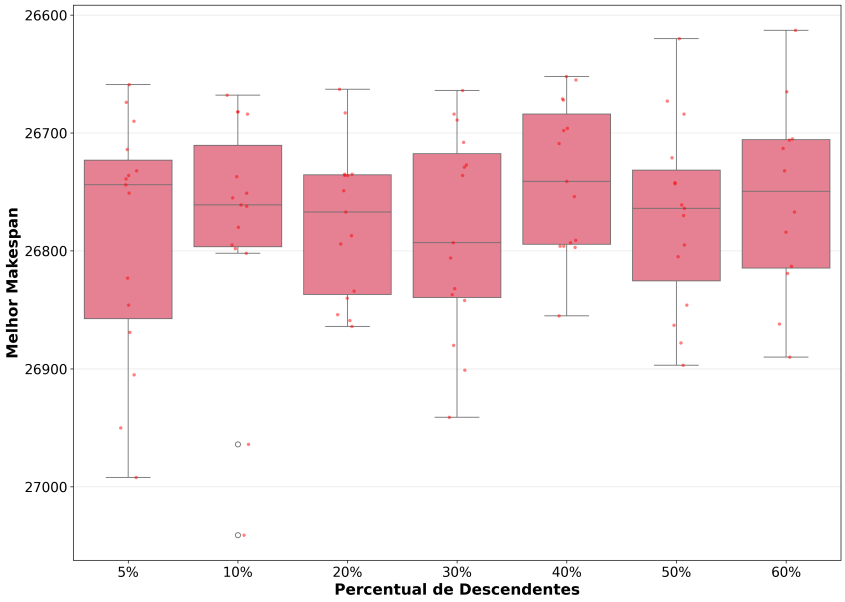
Fonte: O Autor.

6.2.3 Taxa de Introdução de Indivíduos Novos

A taxa de introdução de indivíduos novos por geração controla a proporção da população que é substituída a cada iteração. Esse parâmetro determina o equilíbrio entre a estabilidade populacional e a introdução de diversidade. Em taxas muito baixas, o algoritmo tende à estagnação, e em taxas elevadas, o custo de avaliação cresce rapidamente e a população perde estrutura evolutiva (BRINDLE, 1983; GREFENSTETTE, 1992; ALBA, 2005; EIBEN; SMITH, 2015). Na literatura, a substituição parcial é a abordagem mais comum em problemas de escalonamento do tipo *Flow-Shop*, com valores típicos entre 5% e 30% da população (ARMENTANO; RONCONI, 1999; RUIZ; MAROTO, 2005).

A Figura 30 apresenta o desempenho do algoritmo para diferentes probabilidades de introdução de novos indivíduos. A partir de 30%, o ganho de qualidade, tanto em resultado final quanto em média, torna-se pequeno, enquanto o tempo de execução cresce rapidamente. Observa-se, portanto, que o aumento da taxa de substituição não compensa o custo computacional adicional de avaliação. Além disso, a taxa de 10% já proporciona uma melhoria no valor médio da função objetivo $(26777,47)$, sendo que $26,6\%$ das replicações atingiram o valor de referência de 26.699 unidades.

Figura 30 – Avaliação da Taxa de Introdução de Indivíduos Novos



Fonte: O Autor.

6.2.4 Probabilidade de Cruzamento

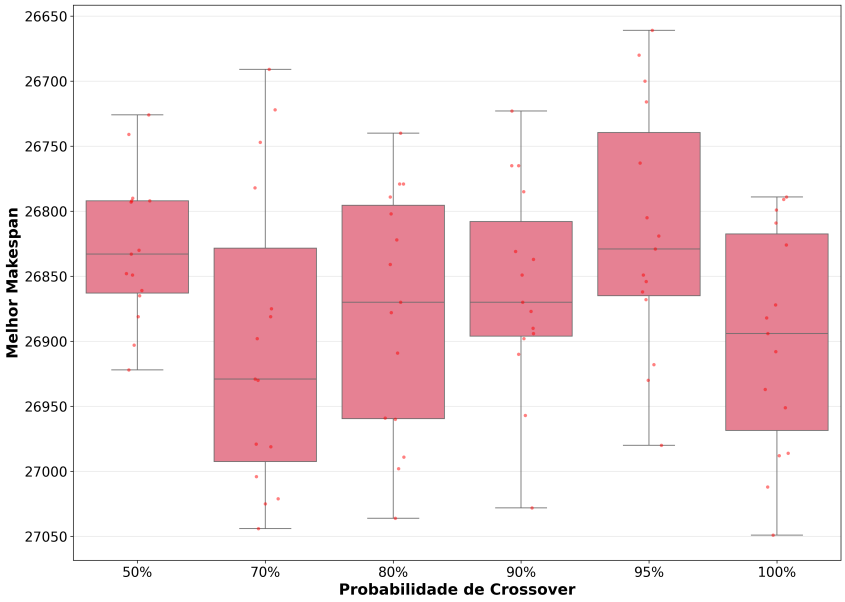
A probabilidade de cruzamento define a frequência com que os indivíduos selecionados geram descendentes por meio da recombinação de seus genes. Esse parâmetro controla o equilíbrio entre a exploração de novas regiões do espaço de busca e a preservação de estruturas promissoras já encontradas. Valores baixos reduzem a capacidade de exploração, tornando o algoritmo excessivamente dependente da mutação, enquanto valores muito altos podem comprometer a diversidade populacional e levar à convergência prematura (HOLLAND, 1992; GOLDBERG, 1989; EIBEN; SMITH, 2015). Na literatura, é comum empregar probabilidades de cruzamento elevadas, geralmente entre 70% e 95%, especialmente em representações permutacionais aplicadas ao problema de *Flow-Shop* (REEVES, 1995; RUIZ; MAROTO, 2005; ALBA, 2005).

A Figura 31 apresenta o comportamento do algoritmo para diferentes probabilidades de cruzamento. Observa-se que a configuração de 95% apresentou o melhor desempenho global, com um valor médio da função objetivo de $26.815,60$. Dessa forma, a taxa de 95% foi adotada como configuração padrão. Neste caso, $26,6\%$ das replicações atingiram o valor de referência UB.

6.2.5 Probabilidade de Mutação

De modo geral, utilizam-se faixas entre 5% e 30% em representações permutacionais aplicadas ao problema de *Flow-Shop* (REEVES, 1995; ARMENTANO; RONCONI, 1999; RUIZ; MAROTO, 2005). A Figura 32 mostra o comportamento do algoritmo para diferentes probabilidades de mutação. Observa-se que valores muito baixos (10%–20%) resultaram em piores médias da função objetivo. Observa-se que o desempenho melhora progressivamente até aproximada-

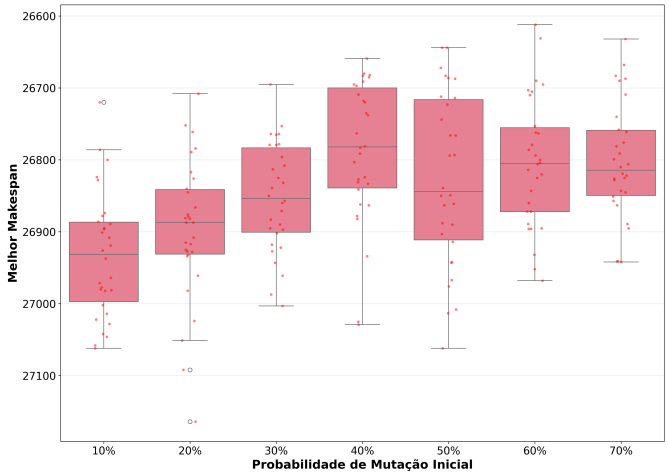
Figura 31 – Avaliação da Probabilidade de Cruzamento



Fonte: O Autor.

mente 40%, ponto em que se obtém o melhor equilíbrio entre qualidade média $(26.788,07)$ e estabilidade dos resultados. A partir desse valor, os resultados apresentam apenas variações pouco significativas. Com essa configuração, $33,3\%$ das replicações atingiram o valor UB de referência.

Figura 32 – Avaliação da Probabilidade de Mutação



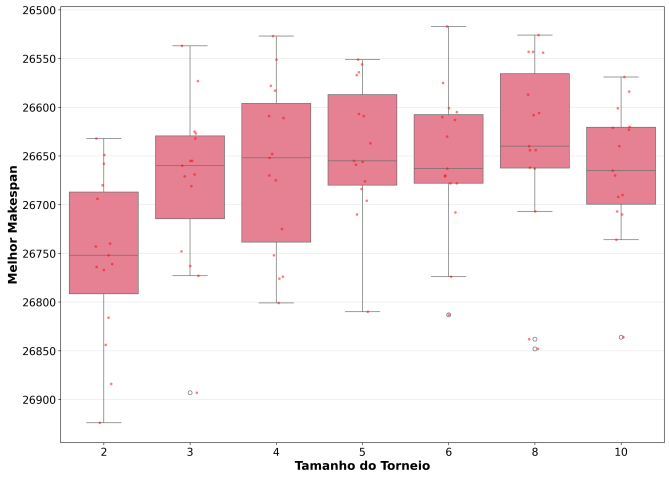
Fonte: O Autor.

6.2.6 Tamanho do Torneio

O tamanho do torneio controla a intensidade da seleção no: torneios maiores favorecem a escolha dos melhores indivíduos, acelerando a convergência, enquanto torneios menores preservam mais diversidade. A literatura recomenda valores reduzidos, geralmente entre 2 e 5 participantes, por oferecerem um bom equilíbrio entre intensificação e diversidade (GOLDBERG, 1989; EIBEN; SMITH, 2015; ALBA, 2005).

A Figura 33 apresenta o impacto desse parâmetro sobre o desempenho do algoritmo. Observa-se que, dentro da faixa avaliada de 2 a 10 participantes, o efeito do tamanho do torneio sobre a qualidade final é moderado, com bons resultados entre 3 e 6 participantes. Em tais casos, todas as configurações atingiram $86,6\%$ de ocorrências no valor de referência UB. Assim, optou-se por um tamanho de torneio igual a 5.

Figura 33 – Desempenho do algoritmo para diferentes tamanhos de torneio



Fonte: O Autor.

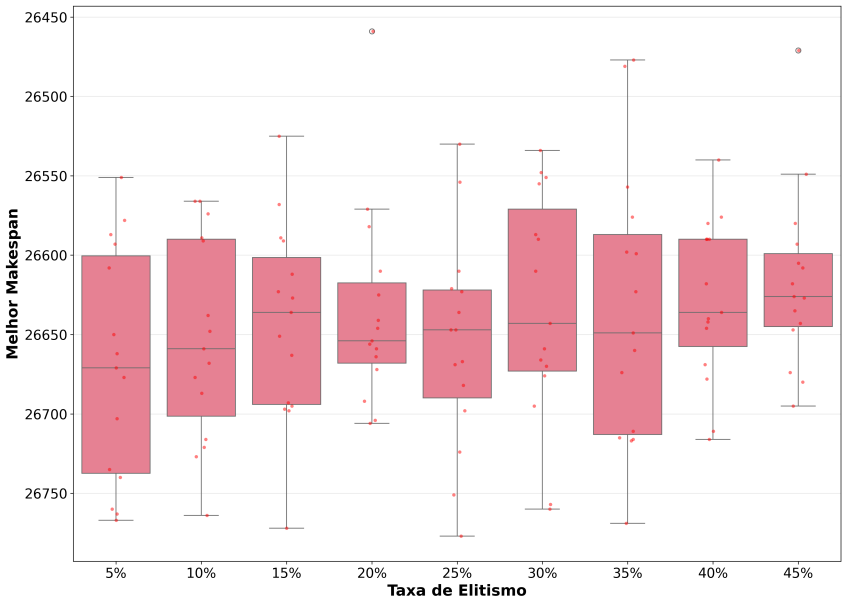
6.2.7 Taxa de Elitismo

Em algoritmos genéticos clássicos, costumam ser adotadas taxas baixas de elitismo, tipicamente entre 1% e 20% da população, para preservar diversidade e reduzir o risco de convergência prematura (GOLDBERG, 1989). Em problemas de escalonamento, essa prática também é comum, pois a qualidade da solução depende da recombinação de sequências promissoras (REEVES, 1995; RUIZ; MAROTO, 2005). Como neste trabalho foi utilizada uma população relativamente grande (900 indivíduos), avaliou-se uma faixa mais ampla de elitismo, entre 5% e 50%.

A Figura 34 mostra que o efeito desse parâmetro sobre a função objetivo. Os resultados apresentaram uma variação de cerca de 50 unidades de tempo, correspondendo a menos de $0,2\%$ em relação ao melhor valor obtido. Observa-se ainda que as melhores médias ocorreram na faixa de 30% a 45% de elitismo. Dentro desse intervalo, a taxa de 45% apresentou um dos

melhores valores médios $(26.616, 73)$ com o segundo menor desvio padrão $(51, 22)$, e 100% das replicações atingindo o valor de referência UB de 26.699 unidades.

Figura 34 – Avaliação da taxa de elitismo



Fonte: O Autor.

6.2.8 Parâmetros Finais do Modelo Sequencial

A partir dos testes, foi definida uma configuração de referência. Essa configuração é apresentada no Quadro 1 e corresponde ao conjunto de parâmetros que apresentou o melhor equilíbrio entre a qualidade das soluções, a estabilidade entre replicações e o custo computacional dentro das faixas testadas.

Quadro 1 – Parâmetros finais do algoritmo genético sequencial

Parâmetro	**Valor adotado**
Tamanho da população	900 indivíduos
Critério de convergência	30.000 gerações sem melhoria
Taxa de introdução de indivíduos novos	10% da população por geração
Probabilidade de cruzamento	95%
Probabilidade de mutação	40%
Tamanho do torneio	5 indivíduos
Taxa de elitismo	45% da população

Fonte: O Autor.

A configuração alcançou a média final da função objetivo de $26.616,2$, com desvio padrão $55,74$. Com essa configuração, 100% da replicações atingiram o valor de UB de referência. Essa configuração foi adotada em todas as subpopulações do modelo em ilhas, servindo como base para todos os testes.

6.3 AVALIAÇÃO DO MODELO EM ILHAS

Esta seção apresenta a análise do modelo em ilhas, avaliando o número de ilhas e o número de gerações entre migrações, uma vez que esses fatores influenciam diretamente o equilíbrio entre diversidade populacional, qualidade das soluções e custo computacional.

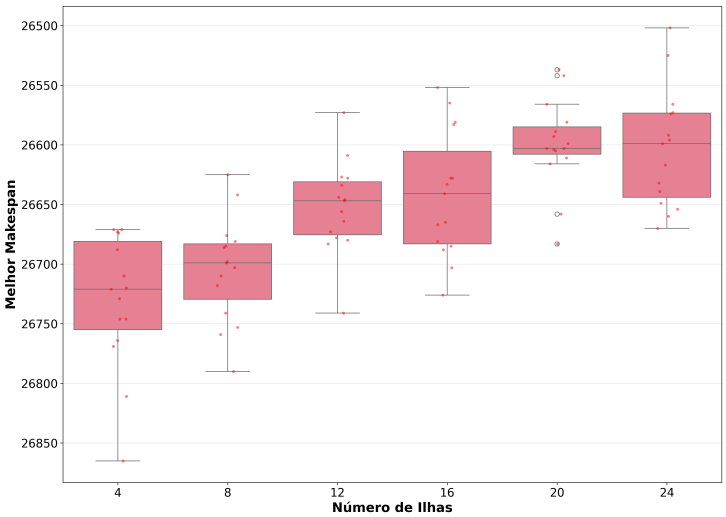
6.3.1 Número de Ilhas

O número de ilhas define o grau de paralelismo do algoritmo e o nível de isolamento evolutivo entre as subpopulações. A divisão da população global em múltiplas ilhas permite que diferentes regiões de busca sejam exploradas simultaneamente, reduzindo o risco de convergência prematura e favorecendo a descoberta de soluções de melhor qualidade (ALBA, 2005; CANTÚ-PAZ, 2000). Na literatura, o número de ilhas adotado varia amplamente conforme o tamanho da população e o ambiente de execução. Estudos clássicos relatam bons resultados com valores entre 4 e 16 ilhas, faixa considerada suficiente para equilibrar diversidade e custo de sincronização (ALBA, 2005; FUKUNAGA; GONG, 2011). Entretanto, trabalhos recentes em arquiteturas de alto desempenho indicam que quantidades maiores de ilhas podem continuar melhorando a qualidade média das soluções (DIAZ-PACE; POMPEO; TUCCI, 2025).

A Figura 35 apresenta o impacto do número de ilhas sobre a qualidade das soluções obtidas. Para esse teste, foi assumido um valor padrão de período para migração de 900 indi-

víduos, migrando 10% da população por vez. Observa-se uma tendência de melhora contínua à medida que o número de ilhas aumenta, indicando que a diversificação promovida pela distribuição populacional exerce um efeito positivo sobre o desempenho global. A média do melhor tempo total de execução reduziu-se de 26.730 com 4 ilhas para 26.599 com 20 ilhas, valor aproximadamente 100 unidades abaixo da melhor métrica reportada por Taillard (1993). Neste caso, o valor de UB de referência foi atingido em 100% da réplicas. Assim, foi estabelecido a configuração de 20 ilhas para os testes restantes.

Figura 35 – Avaliação do número de ilhas



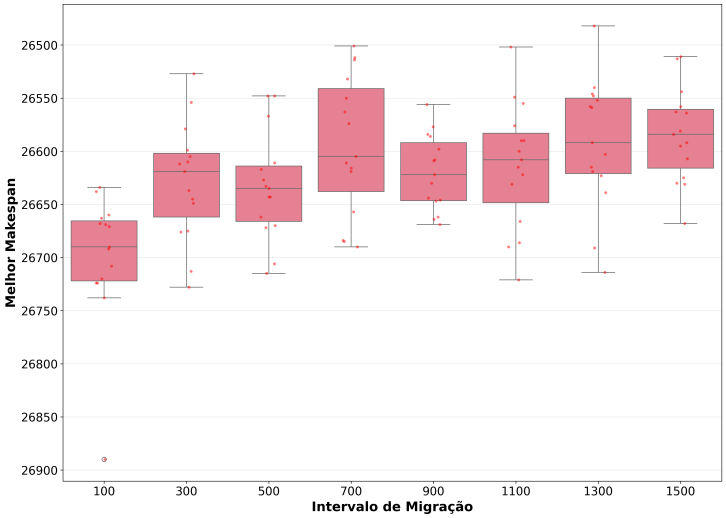
Fonte: O Autor.

6.3.2 Número de Gerações para Migração

O intervalo entre migrações define o número de gerações que cada subpopulação evolui de forma independente antes de trocar indivíduos com suas vizinhas. Intervalos curtos promovem comunicação intensa, acelerando a difusão de boas soluções, porém com risco de reduzir a diversidade global. Intervalos longos, por outro lado, favorecem a exploração local, mas podem retardar a convergência global do sistema (ALBA, 2005; CANTÚ-PAZ, 2000; TOMASSINI, 2005).

A Figura 36 apresenta o impacto da frequência de migração sobre o desempenho do algoritmo. Observa-se que valores muito baixos (100 a 300 gerações) resultam em piores médias, indicando que a comunicação excessiva causa uma convergência prematura. À medida que o intervalo aumenta, as médias melhoram gradualmente, atingindo os melhores resultados na faixa entre 700 e 1100 gerações. A partir de 1300 gerações, o ganho adicional torna-se pequeno. De fato, o valor da função objetivo reflete em menos de 0.2%. Assim, o valor de 700 gerações entre migrações foi adotado como configuração padrão. Com essa configuração, 100% réplicas atigiram o valor de referência UB, atingindo uma média de resultados em $26.595,53$, ou seja, 100 unidades de tempo a menos do que a métrica calculada por Taillard (1993).

Figura 36 – Avaliação da frequência de migração



Fonte: O Autor.

6.3.3 Parâmetros Finais do Modelo em Ilhas

O Quadro 2 apresenta os parâmetros finais do modelo algoritmo genético, incluindo os parâmetros utilizados nos operadores genéticos de cada subpopulação. Essa configuração apresentou um valor médio na função objetivo de $26.595,53$, ultrapassando as métricas propostas por Taillard (1993) em mais de 100 unidades.

Quadro 2 – Configuração final do algoritmo genético paralelo

Parâmetro	**Valor adotado**
Tamanho da população	900 indivíduos
Critério de convergência	30.000 gerações sem melhoria
Taxa de introdução de indivíduos novos	10% da população por geração
Probabilidade de cruzamento	95%
Probabilidade de mutação	40%
Tamanho do torneio	5 indivíduos
Taxa de elitismo	45% da população
Quantidade de Ilhas	20 ilhas
Gerações para Migração	700 gerações

Fonte: O Autor.

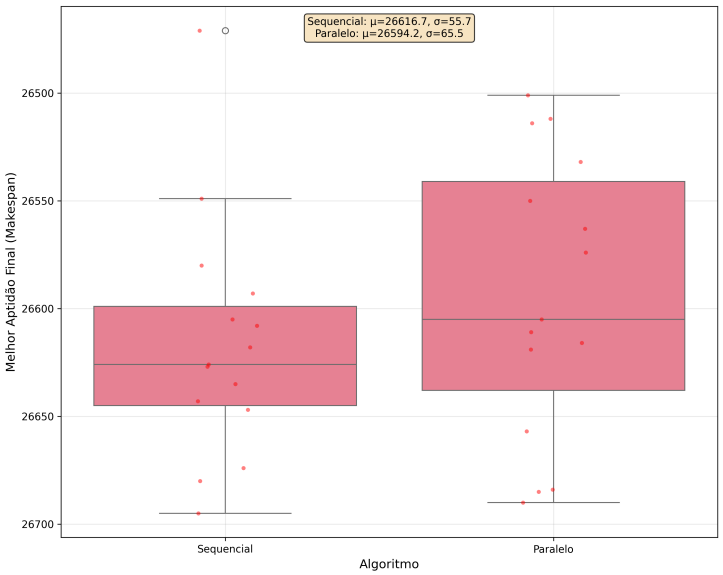
6.4 COMPARAÇÃO ENTRE OS MODELOS SEQUENCIAL E EM ILHAS

A comparação entre o algoritmo genético sequencial e o modelo em ilhas foi realizada inicialmente sem qualquer restrição, permitindo que cada abordagem executasse até a convergência. Posteriormente, foi conduzida uma análise considerando o custo computacional.

6.4.1 Qualidade das soluções

A Figura 37 apresenta os resultados obtidos por ambos os modelos. Observa-se que o modelo em ilhas alcançou, de forma recorrente, valores inferiores da função objetivo, atingindo o limite superior (UB) de 26.699 unidades em todas as execuções realizadas. O algoritmo sequencial também atingiu esse limite, porém sua média permaneceu acima da obtida pelo modelo paralelo. De fato, o modelo em ilhas obteve média final de $26.594,2$, enquanto o algoritmo sequencial apresentou média de $26.616,2$.

Figura 37 – Qualidade das Soluções: Modelo Sequencial vs. Modelo em Ilhas



Fonte: O Autor.

6.4.2 Comparação do Custo Computacional

Para avaliar o desempenho sob condições justas de esforço, Alba (2005), Xiao e Lee (2014), Diaz-Pace, Pompeo e Tucci (2025) detacam que a forma mais adequada de comparar abordagens evolutivas, especialmente em contextos paralelos, é adotar o princípio de orçamento computacional equivalente. Esse método consiste em fixar o número total de avaliações da função objetivo, principal medida de custo computacional em algoritmos genéticos, de modo que todas as abordagens realizem o mesmo esforço de busca. O orçamento adotado corresponde ao maior número de avaliações da função objetivo observado na versão sequencial de $7.607.500$ avaliações.

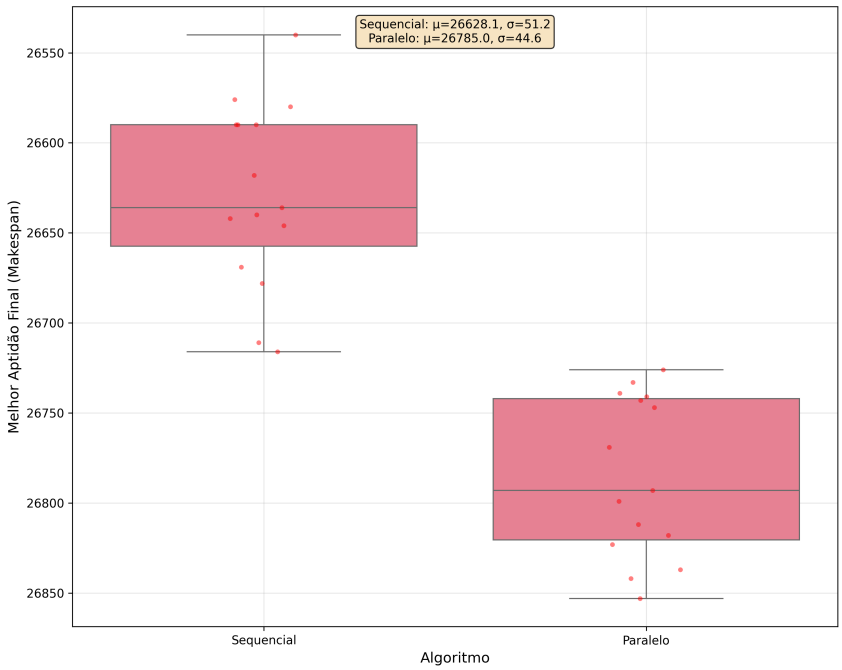
Neste caso, o modelo em ilhas foi configurado com 10 ilhas, correspondentes aos 10 núcleos físicos disponíveis no computador utilizado. Mais especificamente, o computador utilizado nos testes possui um processador *13th Gen Intel® Core™ i7-1365U*, com 10 núcleos físicos e 12 *threads* lógicas, com frequência máxima de 5.2 GHz. O computador também possui 32 GB de memória RAM DDR4, com 928 KB de memória *cache* L1, $6,5$*Megabyte* (MB) de

L2 e 12 de L3. Além disso, o computador possui um disco SSD NVMe de 512 GB. O sistema operacional empregado é um Linux 64 bits Ubuntu (22.04.4 LTS). Não foi possível realizar medições no *cluster*, pois o mesmo encontrava-se em manutenção durante o período dos testes.

Para atingir 7.607.500 avaliações da função objetivo, a versão sequencial levou em média $476,39$ segundos, enquanto o modelo em ilhas levou em média $47,25$ segundos. Nesse caso, o *speedup*[1] é de aproximadamente $10,08$ e a eficiência[2] é de cerca de $1,01$, o que indica que a paralelização utiliza de forma adequada os recursos computacionais disponíveis e que a comunicação entre as ilhas não impacta no desempenho da versão paralela. Nesse caso, o modelo em ilhas torna-se particularmente atrativo, pois permite a avaliação de um número muito maior de indivíduos dentro do mesmo intervalo de tempo. Esse *speedup* superlinear (superior a 10), pode ser decorrente de flutuações nas medições de tempo, que levam a desempenhos superiores ao esperado.

A Figura 38 apresenta a comparação dos valores da função objetivo após 7.607.500 avaliações. Observa-se que o modelo em ilhas, ao ser interrompido antecipadamente, produz soluções com qualidade inferior às obtidas na versão sequencial. Isso ocorre porque, com esse orçamento de avaliações, as subpopulações ainda se encontram na fase inicial de troca de informações, de modo que as melhores soluções ainda não se disseminaram entre as ilhas.

Figura 38 – Comparação do orçamento computacional



Fonte: O Autor.

[1] O *speedup* é a métrica que quantifica o ganho de desempenho da versão paralela em relação à versão sequencial do programa, sendo calculado como $Speedup = \frac{T_{seq}}{T_{par}}$.

[2] A eficiência é a métrica que expressa quão bem os recursos computacionais estão sendo utilizados, sendo calculada como $Eficincia = \frac{Speedup}{p}$, onde p representa o número de núcleos de processamento

7 CONSIDERAÇÕES FINAIS

Este trabalho apresenta um estudo sobre a aplicação de algoritmos genéticos ao problema de Flow-Shop permutacional. Para isso, foi desenvolvida uma implementação sequencial do algoritmo genético e um modelo paralelo baseado em ilhas. Posteriormente, foram realizados testes com o objetivo de definir os parâmetros dos operadores genéticos para ambos os modelos.

A implementação sequencial demonstrou capacidade consistente de atingir e superar o limite superior de 26.699 unidades reportado por Taillard (1993). O algoritmo obteve um valor médio da função objetivo de aproximadamente $26.616,2$ unidades de tempo, com baixa variabilidade entre réplicas. Desta forma, os resultados obtidos indicam que o algoritmo genético proposto é capaz de encontrar soluções com qualidade equiavalentes e até mesmo superiores aos resultados apresentados na literatura.

A partir dos testes realizados com o modelo paralelo baseado em ilhas, verificou-se que a divisão da população em subpopulações, aliada à migração periódica de indivíduos, proporcionou um ganho adicional na qualidade das soluções. Na melhor configuração avaliada, com 20 ilhas e intervalo de migração de 700 gerações, o modelo em ilhas atingiu um valor médio da função objetivo de aproximadamente $26.595,5$ unidades, ou seja, cerca de 100 unidades abaixo do limite superior obtido por Taillard (1993) e cerca de 20 unidades melhor que a média obtida pela versão sequencial. Além disso, os resultados indicam que quantidades maiores de ilhas tendem a melhorar a qualidade média das soluções, como relatado na literatura recente.

A implementação no modelo de ilhas, baseada em comunicação via MPI sobre uma topologia cartesiana bidimensional, apresentou desempenho computacional satisfatório. Nos testes que limitaram o número de avaliações da função objetivo, a versão paralela obteve um *speedup* de aproximadamente 10 vezes, em um computador com 10 núcleos de processamento. Esses resultados indicam que o custo de comunicação associado à migração de indivíduos não afetou de forma significativa o desempenho. Assim, a implementação paralela em ilhas mostra-se atrativa, pois possibilita avaliar um número muito maior de indivíduos em um tempo inferior ou similar a versão sequencial do algoritmo genético.

Uma limitação dos testes foi a utilização de um computador com memória compartilhada para a realização dos mesmos. Isso ocorreu porque o *cluster* disponível encontrava-se em manutenção durante o período de realização dos testes do trabalho. Ainda assim, acredita-se que os *speedups* obtidos seriam semelhantes em um ambiente distribuído, uma vez que o número de gerações entre as migrações é elevado, reduzindo de forma significativa o impacto da comunicação no desempenho geral do modelo paralelo.

7.1 TRABALHOS FUTUROS

Como possíveis trabalhos futuros deste trabalho, destacam-se:

- Realizar os testes de tempo de execução em um ambiente de alto desempenho de *clusters* de computadores, com o objetivo de estimar com maior fidelidade o impacto da comunicação em rede.
- Executar uma busca sistemática dos parâmetros do algoritmo genético, no formato de *grid search* (EIBEN; SMITH, 2015).
- Avaliar o desempenho do modelo em ilhas em outras instâncias clássicas de *Flow-Shop*, explorando diferentes tamanhos e distribuições de tempos de processamento.
- Investigar políticas de migração alternativas, incluíndo topologias ou abordagens adaptativas ou assimétricas, e compará-las com o modelo desenvolvido neste trabalho.
- Explorar versões híbridas do algoritmo genético, combinando o modelo em ilhas com heurísticas construtivas ou estratégias de busca local.

REFERÊNCIAS

AKHSHABI, M.; HADDADNIA, J.; AKHSHABI, M. Solving flow shop scheduling problem using a parallel genetic algorithm. **Procedia Technology**, v. 1, p. 351–355, 2012. ISSN 2212-0173. First World Conference on Innovation and Computer Sciences (INSODE 2011). Disponível em: <https://www.sciencedirect.com/science/article/pii/S2212017312000746>.

ALBA, E. **Parallel metaheuristics: a new class of algorithms**. [S.l.]: John Wiley & Sons, 2005.

ARMENTANO, V. A.; RONCONI, D. P. A genetic algorithm for scheduling in a flowshop problem with setup times. **Computers & Operations Research**, v. 26, n. 3, p. 299–310, 1999.

AWADALLAH, M. A.; ELAZIZ, M. A. Improving genetic algorithm performance by population initialisation with centroid-based solutions. **Computers & Industrial Engineering**, Elsevier, v. 139, p. 106193, 2020.

AZAIEZ, M.-N. *et al.* Two-stage no-wait hybrid flow shop with inter-stage flexibility for operating room scheduling. **Computers Industrial Engineering**, v. 168, p. 108040, 2022. ISSN 0360-8352. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360835222001103>.

BAKER, K. R. **Principles of Sequencing and Scheduling (2nd Edition)**. [S.l.], 2014. Accessed on April 27, 2025. Disponível em: <https://faculty.tuck.dartmouth.edu/images/uploads/faculty/principles-sequencing-scheduling-2e/Notes2e14.pdf>.

BECKER, D. J. *et al.* Beowulf: A parallel workstation for scientific computation. In: **Proceedings of the International Conference on Parallel Processing**. [S.l.: s.n.], 1995.

BLAZEWICZ, J. *et al.* **Scheduling computer and manufacturing processes**. [S.l.]: springer science & Business media, 2013.

BLICKLE, T.; THIELE, L. **A Comparison of Selection Schemes Used in Genetic Algorithms**. [S.l.], 1995. Disponível em: <https://tik-old.ee.ethz.ch/file/6c0e384dceb283cd4301339a895b72b8/TIK-Report11.pdf>.

BOONE, K. *et al.* Combining genetic algorithm with machine learning strategies for designing potent antimicrobial peptides. **BMC Bioinformatics**, v. 22, n. 1, p. 239, 2021.

BRINDLE, A. **Genetic Algorithms for Function Optimization**. Tese (PhD Thesis) — University of Alberta, 1983.

BURKARD, R. E.; ÇELA, E. Quadratic assignment problems. **Handbook of Combinatorial Optimization**, Springer, v. 3, p. 1713–1809, 1998.

BUTENHOF, D. R. **Programming with POSIX Threads**. [S.l.]: Addison-Wesley, 1997. ISBN 978-0-201-63392-4.

BUYYA, R. (Ed.). **High Performance Cluster Computing: Architectures and Systems**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. v. 1. 849 p. ISBN 0-13-013784-7.

BUYYA, R. *et al.* Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, v. 25, n. 6, p. 599–616, 2009. ISSN 0167-739X. Disponível em: <https://doi.org/10.1016/j.future.2008.12.001>.

CALHEIROS, R. N. *et al.* Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. **Software: Practice and Experience**, Wiley, v. 41, n. 1, p. 23–50, 2011.

CANTÚ-PAZ, E. **Efficient and Accurate Parallel Genetic Algorithms**. Boston, MA: Kluwer Academic Publishers, 2000. (Genetic Algorithms and Evolutionary Computation). ISBN 978-0-7923-7221-9, 978-1-4615-4369-5.

CARR, J. **An Introduction to Genetic Algorithms**. 2014. Senior project, Whitman College. Accessed on April 27, 2025. Disponível em: <https://www.whitman.edu/documents/academics/mathematics/2014/carrjk.pdf>.

CASSAR, D. R.; SANTOS, G. G. dos; ZANOTTO, E. D. Designing optical glasses by machine learning coupled with a genetic algorithm. **arXiv preprint arXiv:2008.09187**, 2020. Disponível em: <https://arxiv.org/abs/2008.09187>.

CHANDRAKALA, P. *et al.* Parallel and distributed computing for high-performance applications. In: **ICONNECT-2023**. [S.l.: s.n.], 2023. "Essential for high-performance applications like scientific simulations, data analysis, and artificial intelligence.".

CHAPMAN, B. M. **Using OpenMP: Portable Shared Memory Parallel Programming**. [S.l.]: MIT Press, 2008. ISBN 978-0-262-04239-2.

COHOON, J.; PARIS, W. Genetic placement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 6, n. 6, p. 956–964, 1987.

CONNOR, A. M.; SHAH, A. Resource allocation using metaheuristic search. In: **Proceedings of the Fourth International Conference on Computer Science & Information Technology**. [s.n.], 2014. ArXiv:1605.01855 [cs.NE]. Disponível em: <https://arxiv.org/abs/1605.01855>.

DAUZèRE-PéRèS, S. *et al.* The flexible job shop scheduling problem: A review. **European Journal of Operational Research**, v. 314, n. 2, p. 409–432, 2024.

DAVIS, L. Genetic algorithms and simulated annealing. **Computers & Operations Research**, v. 18, n. 9, p. 919–929, 1991. Disponível em: <https://doi.org/10.1016/0167-6377(91)90014-G>.

DIAZ-PACE, J.; POMPEO, D. D.; TUCCI, M. On the role of search budgets in model-based software refactoring optimization. **Automated Software Engineering**, Springer Science and Business Media LLC, v. 33, n. 1, out. 2025. ISSN 1573-7535. Disponível em: <http://dx.doi.org/10.1007/s10515-025-00564-y>.

DONGARRA, J.; HEMPEL, R.; WALKER, D. The mpi standard for message passing. In: **Proceedings of the Scalable Parallel Libraries Conference**. [S.l.]: IEEE, 1993. p. 159–165.

EBRAHIMI, M. A. *et al.* Solving np hard problems using a new genetic algorithm. **International Journal of Nonlinear Analysis and Applications**, Semnan University, v. 14, n. 1, p. 275–285, 2023. ISSN 2008-6822. Disponível em: <https://ijnaa.semnan.ac.ir/article_6367.html>.

EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. 2nd. ed. Berlin, Heidelberg: Springer, 2015. ISBN 978-3-662-44874-8.

FERREIRA, P.; FARIA, F.; ALVES, R. A genetic algorithm for solving the flexible job shop scheduling problem. **Applied Sciences**, v. 11, n. 10, p. 4425, 2021. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/11/10/4425>.

FOUNDATION, S. **Beowulf Computing Cluster**. 2025. Disponível em: <https://www.spacefoundation.org/space_technology_hal/beowulf-computing-cluster/>.

FUKUNAGA, A. S.; GONG, J. Distributed island-model genetic algorithms using heterogeneous computing resources. In: **Proceedings of IEEE CEC**. [S.l.: s.n.], 2011. p. 820–827. Descreve GA modelo ilha em cluster com comunicação via mensagem entre processadores independentes.

GALLAGHER, K.; SAMBRIDGE, M. Genetic algorithms: A powerful tool for large-scale nonlinear optimization problems. **Computers & Geosciences**, v. 20, n. 7, p. 1229–1236, 1994. ISSN 0098-3004. Disponível em: <https://www.sciencedirect.com/science/article/pii/0098300494900728>.

GANDOMI, A. H. *et al.* **Metaheuristic algorithms**. [S.l.]: Elsevier Waltham, 2013. 1–24 p.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. [S.l.]: W. H. Freeman, 1979.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The complexity of flowshop and jobshop scheduling. **Mathematics of Operations Research**, v. 1, n. 2, p. 117–129, 1976. ISSN 0364-765X. Disponível em: <https://doi.org/10.1287/MOOR.1.2.117>.

GEIST, A. *et al.* Pvm 3 user's guide and reference manual. **Pvm 3 User's Guide And Reference Manual**, 11 1995.

GÖKALP, O.; UĞUR, A. A high-level and adaptive metaheuristic selection algorithm for solving highdimensional bound-constrained continuous optimization problems. **Turkish Journal of Electrical Engineering and Computer Sciences**, v. 28, n. 3, p. 1549–1566, 2020.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Boston: Addison-Wesley, 1989.

GRAMA, A. *et al.* **Introduction to Parallel Computing**. 2nd. ed. [S.l.]: Addison-Wesley, 2003. ISBN 9780201648652.

GREFENSTETTE, J. J. Genetic algorithms for changing environments. **Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature**, p. 137–144, 1992.

GROPP, W.; LUSK, E.; SKJELLUM, A. **Using MPI: Portable Parallel Programming with the Message-Passing Interface**. 2. ed. [S.l.]: MIT Press, 1999. ISBN 978-0262571326.

GROPP, W.; SNIR, M. **MPI: the complete reference. The MPI-2 extensions**. [S.l.]: MIT press, 1998. v. 2.

HANEN, C. Study of a np-hard cyclic scheduling problem: The recurrent job-shop. **European Journal of Operational Research**, v. 72, n. 1, p. 82–101, 1994. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/0377221794903328>.

HARADA, T.; ALBA, E. Parallel genetic algorithms: A useful survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, ago. 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3400031>.

HOLLAND, J. H. Genetic algorithms and adaptation. In: SELFRIDGE, O. G.; RISSLAND, E. L.; ARBIB, M. A. (Ed.). **Adaptive Control of Ill-Defined Systems**. Boston, MA: Springer, 1984. p. 317–333. ISBN 978-1-4684-8943-9. Disponível em: <https://doi.org/10.1007/978-1-4684-8941-5_21>.

____. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**. Cambridge, MA: MIT Press, 1992. Originally published in 1975 by University of Michigan Press. ISBN 978-0-262-58111-0.

HUANG, T.; YIN, H.; HUANG, X. Improved genetic algorithm for multi-threshold optimization in digital pathology image segmentation. **Scientific Reports**, v. 14, n. 1, p. 22454, 2024.

ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Examining the effect of elitism in cellular genetic algorithms using two neighborhood structures. In: **Parallel Problem Solving from Nature – PPSN X**. [S.l.]: Springer, 2008. (Lecture Notes in Computer Science, v. 5199), p. 458–467.

ISLAM, M. T.; RAFA, S. R.; KIBRIA, M. G. Early prediction of heart disease using pca and hybrid genetic algorithm with k-means. **arXiv preprint arXiv:2101.00183**, 2021. Disponível em: <https://arxiv.org/abs/2101.00183>.

IZZO, D.; BISCANI, F.; RUCIńSKI, M. On the impact of the migration topology on the island model. **arXiv preprint arXiv:1004.4541**, 2010.

JAIN, L.; BRANAVAN, S. R. K.; BHATIA, R. S. Termination criteria in evolutionary algorithms: A survey. In: **Proceedings of the International Conference on Evolutionary Computation**. [S.l.: s.n.], 2001. p. 1–6.

JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. **Naval Research Logistics Quarterly**, v. 1, n. 1, p. 61–68, 1954.

KARTHICK, A.; RAMARAJ, E. Comparative study of genetic algorithm, ant colony optimization and particle swarm optimization based job scheduling algorithms for cloud environment. In: **Proceedings of the National Conference on Data Science & Engineering**. Reed Elsevier, 2023. ISBN 978-93-5107-294-2. Disponível em: <https://www.researchgate.net/publication/374259573_Comparative_study_of_Genetic_Algorithm_Ant_Colony_Optimization_and_Particle_Swarm_Optimization_based_job_scheduling_algorithms_for_cloud_environment>.

KATOCH, S.; CHAUHAN, S. S.; KUMAR, V. A review on genetic algorithm: past, present, and future. **Multimedia Tools and Applications**, v. 80, n. 5, p. 8091–8126, 2021. Disponível em: <https://doi.org/10.1007/s11042-020-10139-6>.

KUMAR, M. *et al.* A comprehensive survey for scheduling techniques in cloud computing. **Journal of Network and Computer Applications**, v. 143, p. 1–33, 2019. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804519302036>.

LARSON, S. M. *et al.* Using distributed computing to tackle previously intractable problems in computational biology. **arXiv preprint arXiv:0901.0866**, 2009. “Distributed computing . . . has been able to break previous computational barriers.”.

LEITE, B. S. C. F. **The Job Shop Scheduling Problem: Mixed Integer Programming Models**. 2023. Acessado em 26 de abril de 2025. Disponível em: <https://medium.com/data-science/the-job-shop-scheduling-problem-mixed-integer-programming-models-4bbee83d16ab>.

LIU, Y.; WANG, X.; ZHANG, Y. Improved fitness proportionate selection-based genetic algorithm. In: **Proceedings of the 3rd International Conference on Mechatronics and Information Technology**. [S.l.]: Atlantis Press, 2016. p. 136–140.

LIU, Y.; ZHANG, J.; WANG, Y. An estimation of distribution algorithm for permutation flow-shop scheduling. **Systems**, MDPI, v. 11, n. 8, p. 389, 2023.

LORENA, L. A. N.; LOPES, L. de S. Genetic algorithms applied to computationally difficult set covering problems. **Journal of the Operational Research Society**, v. 48, n. 4, p. 440–445, 1997. ISSN 1476-9360. Disponível em: <https://doi.org/10.1057/palgrave.jors.2600380>.

M., T.; KUMAR, T.; R, N. K. A precedence constrained flow shop scheduling problem with transportation time, breakdown times, and weighted jobs. **Journal of Project Management**, v. 7, p. 229–240, 01 2022.

MALASHIN, I. P. *et al.* Two-stage genetic algorithm for optimization logistics network for groupage delivery. **Applied Sciences**, v. 14, n. 24, p. 12005, 2024.

MANAVI, M.; ZHANG, Y.; CHEN, G. Resource allocation in cloud computing using genetic algorithm and neural network. **arXiv preprint arXiv:2308.11782**, 2023. Disponível em: <https://arxiv.org/abs/2308.11782>.

MARTIN, R. P. *et al.* Effects of communication latency, overhead, and bandwidth in a cluster architecture. In: **Proceedings of the 24th International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 1997. p. 85–96.

MATSUMURA, R. **Busca Local e Otimização**. 2020. Acesso em: 27 abr. 2025. Disponível em: <https://ricardomatsumura.medium.com/busca-local-e-otimiza%C3%A7%C3%A3o-4b69d25eb49e>.

MILLER, B. L.; GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. **Complex Systems**, v. 9, p. 193–212, 1995.

MUCHALSKI, F. J. **Alocação de máquinas virtuais em ambientes de computação em nuvem considerando o compartilhamento de memória**. 64 p. Dissertação (Dissertação (Mestrado em Computação Aplicada)) — Universidade Tecnológica Federal do Paraná, Curitiba, Brasil, 2014.

NAVARRO, C.; HITSCHFELD, N.; MATEU, L. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. **Communications in Computational Physics**, v. 15, p. 285–329, 09 2013.

NGXANDE, M.; MOOROSI, N. Development of beowulf cluster to perform large datasets simulations in educational institutions. **International Journal of Computer Applications**, v. 99, n. 15, p. 29–, 2014. Disponível em: <https://www.researchgate.net/publication/265203371_Development_of_Beowulf_Cluster_to_Perform_Large_Datasets_Simulations_in_Educational_Institutions>.

PAN, Q.-K.; WANG, L.; ZHAO, B.-H. An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. **The International Journal of Advanced Manufacturing Technology**, v. 38, n. 7-8, p. 778–786, 2007. ISSN 0268-3768. Disponível em: <https://doi.org/10.1007/s00170-007-1120-y>.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization: Algorithms and Complexity**. [S.l.]: Dover Publications, 1998.

PARVEEN, S.; ULLAH, H. Review on job-shop and flow-shop scheduling using. **Journal of Mechanical Engineering**, v. 41, 04 2011.

PESCH, E. **The Job Shop Scheduling Problem: Conventional and new Solution Techniques**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994. 138–138 p. ISBN 978-3-642-78910-6.

PINEDO, M. L. **Scheduling: Theory, Algorithms, and Systems**. 5th. ed. [S.l.]: Springer, 2016. ISBN 978-1-4939-3102-0.

RAGGER, H. *et al.* Architectures and message-passing algorithms for cluster computing. **Parallel Computing**, v. 25, n. 9–10, p. 1153–1176, 1999. Analisa arquitetura de clusters e algoritmos de passagem de mensagem e seu impacto em desempenho.

REEVES, C. R. A genetic algorithm for flowshop sequencing. **Computers & Operations Research**, Elsevier, v. 22, n. 1, p. 5–13, 1995.

RITT, M.; ROSSIT, D. Effective heuristics for permutation and non-permutation flow shop scheduling with missing operations. **Computers Operations Research**, v. 170, 07 2024.

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics. **European Journal of Operational Research**, v. 165, n. 2, p. 479–494, 2005.

SAID, G. A. E.-N. A.; MAHMOUD, A. M.; EL-HORBATY, E.-S. M. A comparative study of meta-heuristic algorithms for solving quadratic assignment problem. **International Journal of Advanced Computer Science and Applications (IJACSA)**, v. 5, n. 1, p. 1–6, 2014. Disponível em: <https://doi.org/10.14569/IJACSA.2014.050101>.

SASAKI, S. *et al.* Using genetic algorithms to optimise current and future health planning - the example of ambulance locations. **International Journal of Health Geographics**, v. 9, n. 1, p. 4, 2010. ISSN 1476-072X. Disponível em: <https://doi.org/10.1186/1476-072X-9-4>.

ŠEDA, M. Mathematical models of flow shop and job shop scheduling problems. **World Academy of Science, Engineering and Technology**, Citeseer, v. 1, n. 31, p. 122–127, 2007.

SILVA, C. *et al.* A logistic process scheduling problem: Genetic algorithms or ant colony optimization? **IFAC Proceedings Volumes**, v. 38, n. 1, p. 206–211, 2005. ISSN 1474-6670. 16th IFAC World Congress. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474667016375309>.

SINGH, A. K.; SINGH, H.; VARSNEY, M. The significance and diversity of task scheduling methods in cloud computing platforms. **International Journal of Intelligent Systems and Applications in Engineering**, v. 12, n. 14s, p. 644 –, Feb. 2024. Disponível em: <https://ijisae.org/index.php/IJISAE/article/view/4727>.

SISSODIA, R.; RAUTHAN, M. S.; BARTHWAL, V. A multi-objective optimization scheduling method based on the genetic algorithm in cloud computing. **International Journal of Cloud Applications and Computing (IJCAC)**, v. 12, n. 1, p. 1–21, 2022. Disponível em: <https://doi.org/10.4018/IJCAC.305217>.

SKORPIL, V.; OUJEZSKY, V. Parallel genetic algorithms' implementation using a scalable concurrent operation in python. **Sensors**, v. 22, n. 6, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/6/2389>.

SKORPIL, V.; OUJEZSKý, V.; TULEJA, M. Programming reconfigurable heterogeneous computing clusters. **arXiv preprint arXiv:2007.12345**, 2020. MPI tornou-se o método dominante em clusters HPC, sendo adotado como padrão de fato para aplicações científicas. Disponível em: <https://arxiv.org/abs/2007.12345>.

SONG, X.; GAO, L.; WANG, J. Job scheduling based on ant colony optimization in cloud computing. In: **2011 International Conference on Computer Science and Service System (CSSS)**. [S.l.: s.n.], 2011. p. 3309–3312.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, v. 64, n. 2, p. 278–285, 1993. Disponível em: <http://mistic.heig-vd.ch/taillard/articles.dir/Taillard1993EJOR.pdf>.

TOMASSINI, M. Spatially structured evolutionary algorithms. In: ALBA, E. (Ed.). **Parallel Metaheuristics: A New Class of Algorithms**. [S.l.]: John Wiley & Sons, 2005. p. 51–84.

WANG, Y.; SUN, Y.; SUN, Y. Task scheduling algorithm in cloud computing based on fairness load balance and minimum completion time. In: **Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering**. [S.l.]: Atlantis Press, 2015. p. 195–204.

WOEGINGER, G. J. Exact algorithms for np-hard problems: A survey. In: ____. **Combinatorial Optimization — Eureka, You Shrink!: Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 185–207. ISBN 978-3-540-36478-8. Disponível em: <https://doi.org/10.1007/3-540-36478-1_17>.

XIAO, H.; LEE, L. H. Simulation optimization using genetic algorithms with optimal computing budget allocation. **SIMULATION**, v. 90, p. 1146–1157, 10 2014.

XIAO, Y.-Y. *et al.* Permutation flow shop scheduling with order acceptance and weighted tardiness. **Applied Mathematics and Computation**, v. 218, n. 15, p. 7911–7926, 2012. ISSN 0096-3003. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0096300312001245>.

XIONG, H. *et al.* A survey of job shop scheduling problem: The types and models. **Computers & Operations Research**, v. 142, p. 105731, 2022. ISSN 0305-0548. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305054822000338>.

YANG, X.-S. **Nature-Inspired Metaheuristic Algorithms**. 1st. ed. Frome, UK: Luniver Press, 2008. ISBN 9781905986102.

YOUSEFF, L.; BUTRICO, M.; SILVA, D. D. Toward a unified ontology of cloud computing. In: IEEE. **2008 Grid Computing Environments Workshop**. [S.l.], 2008. p. 1–10.

ZAEFFERER, M. Permutation distance measures in ego. In: **Proceedings of the 16th International Conference on Parallel Problem Solving from Nature (PPSN'14)**. [S.l.: s.n.], 2014.

ZHANG, Z.; WANG, J. A novel approach for signal processing. **Circuits, Systems, and Signal Processing**, v. 38, n. 12, p. 5612–5623, 2019. Disponível em: <https://link.springer.com/article/10.1007/s00034-019-01037-w>.

ZHOU, H.; GRACIA, J.; SCHNEIDER, R. Mpi collectives for multi-core clusters: Optimized performance of the hybrid mpi+mpi parallel codes. **arXiv preprint arXiv:2007.06892**, 2020. Aborda operações coletivas (allgather, broadcast) e sincronização entre processos em clusters multicore com MPI. Disponível em: <https://arxiv.org/abs/2007.06892>.

ZOUNMEVO, J. A. *et al.* Using mpi in high-performance computing services. In: **Proceedings of the ACM Symposium on High Performance Computing Services**. [s.n.], 2018. MPI é um dos modelos de programação HPC mais portáveis, com implementações otimizadas por plataforma. Disponível em: <https://www.mcs.anl.gov/papers/P5482-1215.pdf>.

APÊNDICE A – DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Durante a preparação deste trabalho, o autor utilizou a ferramenta ChatGPT[1] versão 5 para revisão ortográfica e ajustes de coesão textual. Após o uso da ferramenta, o autor revisou e editou o conteúdo em conformidade com o método científico e assume total responsabilidade pelo conteúdo da publicação.

[1] https://chatgpt.com/