UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Celso Emiliano Borges Cintra

Problemas de Escalonamento e Meta-heurísticas de IA

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Celso Emiliano Borges Cintra

Problemas de Escalonamento e Meta-heurísticas de IA

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Profª. Dra. Márcia Aparecida Fernandes

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2024

Celso Emiliano Borges Cintra

Problemas de Escalonamento e Meta-heurísticas de IA

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 07 de fevereiro de 2024:

Prof[a]. Dra. Márcia Aparecida Fernandes
Orientadora
Universidade Federal de Uberlândia

Banca Examinadora:

Prof. Dr. Alexsandro Santos Soares
Universidade Federal de Uberlândia

Prof. Dr. Paulo Henrique Ribeiro Gabriel
Universidade Federal de Uberlândia

Uberlândia, Brasil
2024

Resumo

Os Algoritmos Evolutivos são uma classe de algoritmos de busca e otimização que se baseiam na teoria da evolução, através de processos iterativos, com a utilização de populações candidatas e buscam encontrar solução de um problema. O problema de programação de FJSP (*Flexible Job Shop Scheduling Problem*) é um problema de otimização combinatória NP-difícil, em que um conjunto de Jobs compostos por tarefas são escalonados, e este problema que tem aplicações no mundo real. Devido à sua complexidade e importância, muita atenção tem sido dada para resolver este problema. Os problemas FJSP consistem em dois subproblemas principais, que são, a atribuição de operações às máquinas e o sequenciamento de operações. Em problemas de escalonamento de difícil solução, os métodos tradicionais não são capazes de fornecer resultados satisfatórios. Além disso, muitas vezes não existe um algoritmo exato para solução do problema. Assim, deve-se buscar técnicas mais eficientes ou capazes de retornar resultados de forma mais rápida, e neste contexto a aplicação de meta-heurísticas tem obtido os melhores resultados. Este trabalho tem como um dos objetivos principais a alteração de um algoritmo já criado por alunos do Doutorado e Mestrado da UFU, consiste em um híbrido baseado no *Estimation of distribution algorithm* (EDA) e aplicado em *Heterogeneous Computing Scheduling Problem* (HCSP). Para testes e obtenção de resultados, o algoritmo trabalhou com *DataSets* já conhecidos, propostos por Kacem, Hammadi e Borne (2002). Foram analisados os resultados obtidos e comparados com outros trabalhos que utilizam meta-heurísticas diversas, e assim, verificou-se que o algoritmo, incluindo as alterações propostas e executadas, apresentou resultados satisfatórios para os problemas avaliados.

Palavras-chave: meta-heurística, otimização, escalonamento, jobs.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AE	Algoritmo Evolutivo
AEMO	Algoritmos Evolutivos Multiobjetivo
EDA	Estimation of distribution algorithm
EDAh	EDA híbrido
ETC	Matriz de Tempos
FJSP	Flexible Job Shop Scheduling Problem
GA	Genetic Algorithm
HCSP	Heterogeneous Computing Scheduling Problem
SI	Swarm Intelligence
JSP	Job Shop Scheduling Problem
MOFJSP	Multi-Objective Flexible Job Shop Scheduling Problem
MP	Matriz de Probabilidade
PSO	Particle Swarm Optimization

Sumário

1 Introdução

Computação evolutiva refere-se a uma classe de algoritmos de otimização inspirados no processo evolutivo biológico. Essa abordagem envolve a aplicação de mecanismos como seleção natural, cruzamento (crossover), mutação e hereditariedade, para encontrar soluções aproximadas ou ótimas para problemas complexos. O algoritmo evolutivo (AE) é uma subdivisão da computação evolutiva e está inserido no conjunto de algoritmos que compõem a busca estocástica geral, conforme Vikhar (2016). Meta-heurísticas são estratégias de alto nível desenvolvidas para empregar heurísticas de maneira inteligente, buscando eficientemente soluções quase ótimas para desafios complexos de otimização, como abordado por Maier et al. (2019).

Numa visão geral, os problemas de escalonamento são formulados como problemas de otimização e são amplamente estudados, em especial devido a sua importância e aplicabilidade. Eles podem ser encontrados em diversas áreas, seja na indústria de fabricação, nos serviços de logística, e com ênfase na solução de problemas computacionais distribuídos. Para estes problemas de escalonamento, meta-heurísticas podem e devem ser utilizadas, pois são as que apresentam resultados mais satisfatórios, principalmente por serem capazes de lidar melhor com a natureza combinatorial destes problemas do que os métodos tradicionais (VARELA, 2007).

Neste trabalho, são tratados os problemas de escalonamento FJSP (*Flexible Job Shop Scheduling Problem*) que consistem em dois subproblemas principais, que são a atribuição de operações às máquinas e o sequenciamento de operações. Conforme Gao et al. (2019), atribuição de máquinas é selecionar um equipamento de um conjunto candidato para cada operação, enquanto sequenciamento de operações entende-se por escalonar todas as operações de cada *Job* de acordo com a ordem previamente estabelecida. Como o FJSP foi provado ser um problema complexo de ser resolvido (NP-difícil), métodos tradicionais de otimização matemática não são capazes de lidar com esses problemas em um período de tempo razoável, sendo necessário recorrer a Algoritmos Evolutivos (EA) e Inteligência de enxame (*Swarm Intelligence*).

Dentre as diversas meta-heurísticas existentes na literatura, devido ao bom resultado apresentado na solução de problemas do tipo HCSP (*Heterogeneous Computing Scheduling Problem*), para este trabalho foi utilizado EDA (*Estimation Distribution Algorithms*). EDA são algoritmos evolutivos, mas que diferenciam-se de outros algoritmos evolutivos tradicionais por não utilizar cruzamentos na geração de novos indivíduos.

Em algoritmos EDA, ao invés de cruzamento, utilizam estimativa de distribuição de probabilidades para gerá-los, tendo o modelo probabilístico como ponto central, e que

por sua vez é o responsável pela geração dos novos indivíduos, como é mostrado em trabalhos como o de Carvalho (2021).

1.1 Objetivo

O objetivo deste trabalho de conclusão de curso é aplicar um algoritmo híbrido baseado na meta-heurística EDA em problema de escalonamento do tipo *Flexible Job Shop Scheduling Problem* (FJSP). O algoritmo mencionado foi desenvolvido para o problema *Heterogeneous Computing Scheduling Problem* (HCSP) e obteve resultados compatíveis com a literatura. Entretanto, estes problemas têm características diferentes e, por isso, alterações foram necessárias para atender ao FJSP.

Assim, os objetivos específicos foram:

- Compreender o algoritmo híbrido baseado em EDA aplicado ao *Heterogeneous Computing Scheduling Problem* (HCSP), a fim de determinar as alterações necessárias.
- Adaptar o algoritmo híbrido para aplicação ao *Flexible Job Shop Scheduling Problem* (FJSP), preservando ao máximo as especificações do algoritmo a fim de também obter bons resultados.
- Avaliar o algoritmo adaptado em *benchmarks* conhecidos para o *Flexible Job Shop Scheduling Problem* (FJSP).
- Comparar e analisar resultados em relação a trabalhos que utilizam outras meta-heurísticas.

1.2 Justificativa

Geralmente, em problemas de escalonamento de difícil solução, os métodos tradicionais não são capazes de fornecer resultados satisfatórios. Além disso, muitas vezes não existe um algoritmo exato para solução do problema. Assim, deve-se buscar técnicas mais eficientes ou capazes de retornar bons resultados de forma mais rápida, e neste contexto, a aplicação de meta-heurísticas tem obtido os melhores resultados.

Neste sentido, como a meta-heurística EDA (*Estimation of distribution algorithm*), em especial uma versão híbrida, aplicada na solução de problemas de escalonamentos, como HCSP (*Heterogeneous Computing Scheduling Problem*), obteve bons resultados anteriormente, buscou-se aproveitar deste algoritmo neste trabalho, efetuando-se modificações de forma a adaptá-lo à busca de soluções de problemas do tipo FJSP (*Flexible Job Shop Scheduling Problem*), com a expectativa de que ele também conseguisse bons resultados, tendo em vista que ambos tratam de problemas de escalonamento, porém com

suas particularidades, torna-se viável contornar questões pontuais relativas às restrições presentes no FJSP para se alcançar o objetivo desejado.

1.3 Organização do Trabalho

Este trabalho de TCC está dividido, a partir deste ponto, nos seguintes capítulos:

- Capítulo 2, em que é apresentado o referencial teórico envolvido no trabalho como a meta-heurística EDA, além da definição dos problemas tratados. Ainda no Capítulo 2, tem-se a revisão da literatura, onde se discutem algumas meta-heurísticas utilizadas para resolver problemas FJSP.
- No Capítulo 3 é apresentado o desenvolvimento da adaptação do algoritmo.
- No Capítulo 4 são apresentados os resultados e a discussão.
- Por fim, no Capítulo 5, a conclusão.

2 Referencial Teórico

Neste capítulo são abordados os principais fundamentos teóricos necessários ao desenvolvimento deste TCC, como conceitos e definições relativos à meta-heurística utilizada, neste caso o EDA, além da definição dos problemas de escalonamento apresentados, como HCSP e FJSP. Também são apresentados os trabalhos relacionados e os que foram utilizados como referência.

2.1 Algoritmos Evolutivos

Os Algoritmos Evolutivos (AE) são uma classe de algoritmos de busca e otimização que se baseiam na teoria da evolução, através de processos iterativos, com a utilização de populações candidatas e buscam encontrar solução de um problema. Os AE são especialmente úteis em problemas complexos, nos quais as abordagens tradicionais podem enfrentar dificuldades, proporcionando uma metodologia flexível e poderosa para explorar espaços de solução extensos e multidimensionais (VIKHAR, 2016).

Um AE é um método iterativo em que cada iteração é denominada geração, e ele emprega operadores estocásticos em um conjunto de indivíduos (a população P) com o objetivo de aprimorar sua aptidão, medida relacionada à função objetivo. Cada indivíduo na população representa uma versão codificada de uma solução tentativa para o problema. A população inicial é gerada por meio de um método aleatório ou pela utilização de uma heurística específica para o problema em questão. Uma função de avaliação atribui um valor de aptidão a cada indivíduo, indicando sua adequação ao problema. De maneira iterativa, a aplicação probabilística de operadores de variação, como a recombinação de partes de dois indivíduos ou mudanças aleatórias (mutações) em seus conteúdos, é conduzida por uma técnica de seleção dos melhores, visando soluções tentativas de maior qualidade (VIKHAR, 2016).

2.2 *Estimation of Distribution Algorithms*

Os algoritmos EDA (*Estimation Distribution Algorithms*) são algoritmos evolutivos, mas diferenciam-se de outros algoritmos evolutivos tradicionais, como por exemplo Algoritmos Genéticos, por não utilizar cruzamentos para geração de novos indivíduos. Ao invés disso, utilizam estimativa de distribuição de probabilidades para gerá-los.

O EDA tem o modelo probabilístico como ponto central, que consiste na modelagem de como estimar as probabilidades de distribuição. Esse modelo é o responsável

por gerar os novos indivíduos, fazendo a escolha de cada elemento do indivíduo de forma probabilística. Dessa forma, sua estrutura e construção estão associadas à definição do problema e à representação da solução. (CARVALHO, 2021)

O algoritmo EDA, é composto pelas seguintes etapas:

- A população é inicializada com um tamanho *[tam]*.
- Avalia-se a População que foi inicializada.
- Executa-se um laço de interação (repetindo-se até alcançar o critério de parada).

É dentro deste laço que a população evolui, onde ocorre a busca, e se dá na seguinte maneira:

- Seleção dos indivíduos para construção do modelo probabilístico.
- Construção do modelo probabilístico, com uso de uma matriz de probabilidades.
- Geração de novos indivíduos.
- Substituição dos indivíduos, selecionados no início do laço, pelos gerados aqui.
- Avaliação dos indivíduos por uma função de aptidão.

Esse laço de iteração deve ser executado até que o critério de parada, definido, seja alcançado. Como exemplo de critério de parada: atingir uma quantidade pré-definida de gerações.

Algoritmo 1 EDA

Entrada: tamPop

```
1: pop ← inicializarPopulacao(tamPop)
2: avaliar(pop)
3: enquanto não atingir critério de parada faça
4:     es ← selecionar(pop, tamPm)
5:     mp ← construirModeloProbabilistico(es)
6:     inds ← criarNovosIndividuos(mp, tamPop − tamPm)
7:     pop ← substituirIndividuos(pop, es, inds)
8:     avaliar(inds)
9: fim enquanto
```

2.3 Problemas de HCSP

Por sistema de computação heterogêneo podemos entender como um conjunto coordenado de elementos de processamento, que neste contexto são chamados de recursos,

processadores ou simplesmente máquinas, e que estão interconectados através de uma rede. Essa característica heterogênea do HCSP (*Heterogeneous Computing Scheduling Problem*) está diretamente ligada à variedade nas capacidades computacionais dos diversos recursos computacionais disponíveis neste sistema. Sendo assim, em um ambiente heterogêneo, as máquinas ou processadores possuem diferentes níveis de desempenho, potência de processamento dentre outras características relacionadas ao poder computacional. E, essa heterogeneidade surge devido às diferenças nas arquiteturas de hardware, velocidade de *clock* da CPU, tamanhos de memória, dentre outros.

Além da questão de heterogeneidade existente nas várias máquinas desse sistema, há outro ponto de relevância, que é o conjunto de tarefas com requisitos computacionais variáveis a serem executados no sistema. Por tarefa entendesse como a unidade de carga de trabalho atômica, que não pode ser dividida em pedaços menores nem interrompida após o início de sua execução. E, esse tempo de execução de cada tarefa individual varia de uma máquina para outra.

O objetivo, ou desafio principal, no contexto dos problemas de HCSP, é encontrar uma atribuição eficiente de tarefas ou operações a esses recursos heterogêneos disponíveis, levando em consideração as disparidades existentes em suas capacidades. E isso adiciona complexidade ao problema, já que não basta simplesmente distribuir tarefas, mas sim otimizar essa distribuição com base nas características individuais e específicas de cada recurso. Sendo a métrica de otimização geralmente relacionada à eficiência global do sistema, e também a utilização eficaz dos recursos com uma qualidade de serviço adequada.

		Máquinas			
		Maq.0	Maq.1	Maq.2	Maq.3
Tarefas	Oper.0	3	1	1	2
	Oper.1	2	4	2	1
	Oper.2	2	1	3	5
	Oper.3	4	2	5	3
	Oper.4	2	3	1	4
	Oper.5	5	1	4	2

Figura 1 – Tabela ETC para HCSP. Fonte: Autoria Própria

Partindo-se do pressuposto de que o tempo de execução de cada tarefa em cada uma das máquinas seja conhecido, e representado por exemplo através de uma matriz ETC (Figura 1), o HCSP propõe então encontrar uma atribuição de tarefa/operações às máquinas (Figura 2) que irá otimizar alguma métrica de qualidade. Assim, o escalonador pode concentrar-se na otimização do makespan, que é um critério de otimização bem conhecido, que é definido como o período de tempo entre o início da primeira tarefa e a conclusão da última tarefa. Dessa forma, o makespan é uma medida da produtividade

(*throughput*) de um sistema de computação.

Operações	0	1	2	3	4	5
Máquina selecionada	2	3	0	1	2	3

Figura 2 – Atribuição HCSP. Fonte: Autoria Própria

O HCSP não leva em consideração as interdependências entre as tarefas. Sendo que o problema formulado, parte do pressuposto de que todas as tarefas ou operações podem ser realizadas de forma independente, e sem levar em conta a sequência de suas execuções. Sendo uma versão simplificada do problema de escalonamento mais abrangente, que em outros casos, levaria em consideração as interdependências entre as tarefas, nesse modelo de tarefas independentes destaca-se em ambientes de computação distribuída. Aplicações que envolvem tarefas independentes são comuns em diversas áreas de pesquisa científica, e também em infraestruturas compartilhadas, onde diversos usuários submetem tarefas para execução (MASSOBRIO; DORRONSORO; NESMACHNOW, 2018).

2	3	0	1	2	3

Figura 3 – Indivíduo no HCSP.

2.4 Problemas de FJSP

Os problemas FJSP (*Flexible Job Shop Scheduling Problem*) consistem em dois subproblemas principais, que são, a atribuição de operações às máquinas e o sequenciamento de operações.

Conforme Gao et al. (2019), atribuição de máquinas é selecionar um equipamento de um conjunto candidato para cada operação, enquanto sequenciamento de operações entende-se por escalonar todas as operações de cada *Job* de acordo com a ordem previamente estabelecida.

Como a maioria dos problemas FJSP são complexos de serem resolvidos (NP-difícil), métodos tradicionais de otimização matemática não são capazes de lidar com esses problemas em um período de tempo razoável. É necessário desta forma, recorrer a Algoritmos Evolutivos (EA) e Inteligência de enxame (SI) (WANG et al., 2008).

2.4.1 Modelo matemático da FJSP

O FJSP consiste em um conjunto de máquinas e em um conjunto de *Jobs*, tal que cada *Job* é composto por uma sequência de operações. O objetivo é atribuir todas as operações dos *Jobs* às máquinas, respeitando sempre essa ordem sequencial. Cada

operação da sequência requer e é processada por uma única máquina, e por sua vez cada máquina manipula apenas uma operação por vez. Sendo assim, a operação a ser executada na sequência, deve sempre aguardar o final da execução da operação em andamento, para posteriormente ser atribuída a alguma máquina que esteja disponível para executá-la.

Notações e suposições necessárias na formulação de um FJSP multiobjetivo, conforme Gao et al. (2019), são as seguintes:

1) $J = \{J_i\}, 1 \leq i \leq n$, é um conjunto de n *Jobs* a serem escalonados, q_i denota o número total de operações do *Job* $J = \{1, 2, \ldots, n\}$.

2) $M = \{M_k\}, 1 \leq k \leq m$, é um conjunto de m máquinas.

3) *Job* J_i consiste em uma sequência predeterminada de operações. Assim $O_{i,h}$ refere-se à operação h de J_i.

4) Cada operação $O_{i,h}$ pode ser processada, sem interrupção, em uma máquina candidata $M(O_{i,h})$. $P_{i,h,k}$ denota o tempo de processamento de $O_{i,h}$ na máquina M_k.

5) Variáveis de decisão

$$x_{i,h,k} = \begin{cases} 1, \textit{se a máquina k for selecionada para operação } O_{i,h} \\ 0, \textit{caso contrário} \end{cases}$$

onde o tempo de conclusão da operação $O_{i,h}$ é denotado como $c_{i,h}$.

Existem, em literaturas publicadas, muitos objetivos para FJSP, incluindo tempo de conclusão, tempo de fluxo, carga de trabalho da máquina, data de vencimento, custo e consumo de energia. Esses objetivos são formulados da seguinte forma:

- O tempo máximo de conclusão de todos os *Jobs* chamados *Makespan*: Que é o critério utilizado para comparar os resultados da heurística, e representa o menor valor máximo das execuções.
- $C_{max} = max_{(1 \leq i \leq n)} c_i$, Onde c_i é o tempo de conclusão do *Job* J_i.
- O tempo total de fluxo, $C_{Flow} = \Sigma_{(1 \leq i \leq n)} c_i$, Onde c_i é o tempo de conclusão do *Job* J_i.
- A carga de trabalho máxima da máquina, $W_{max} = max_{(1 \leq j \leq m)} w_j$, onde w_j é a carga de trabalho da máquina M_j.
- A carga de trabalho total da máquina, $W_{Total} = \Sigma_{(1 \leq j \leq m)} w_j$, onde w_j é a carga de trabalho da máquina M_j.
- Minimizar a antecipação ou atraso, $\triangle_i = |c_i - d_i|$, onde d_i é a data de vencimento J_i.

2.4.2 FJSP considerando restrições

No mundo real, são várias as restrições a serem consideradas para se resolver o FJSP, como por exemplo: tempo de processamento incerto ou estocástico, quebra ou interrupção de máquina, restrição de recursos, tempo de configuração, tempo de transporte da operação, inserção dinâmica de novo trabalho, sobreposição de operação, manutenção preventiva, custo ou consumo de energia.

Essas restrições aumentam a dificuldade em se obter soluções de qualidade para FJSP. E de modo geral, pesquisadores consideram uma ou mais dessas restrições ao se resolver o FJSP, conforme Gao et al. (2019).

Neste trabalho não foram consideradas as restrições acima citadas, já que o intuito principal foi verificar a possibilidade de adaptação do algoritmo EDA híbrido aos problemas FJSP na busca de bons resultados e soluções ótimas, considerando um único objetivo, que é o makespan.

2.5 Trabalhos Correlatos

FJSP é um problema de otimização combinatória NP-difícil, que tem diversas aplicações no mundo real. Como abordado por Xie Liang Gao (2019), devido à sua complexidade e importância, muita atenção tem sido dada para resolver este problema. Neste estudo, apresentado pelo referido artigo, os métodos e soluções existentes para o FJSP na literatura recente são classificados em algoritmos exatos, heurísticos e meta-heurísticas, e são revisados de forma abrangente. Além disso, aplicações FJSP do mundo real também são apresentadas, além de se analisar as tendências de desenvolvimento da indústria de transformação, e de futuras oportunidades de pesquisa de FJSP que é uma extensão do clássico *Job Shop Scheduling Problem* (JSP). E, uma das técnicas utilizadas para resolver este tipo de problema é Algoritmo Genético.

Em Driss e Laggoun (2015) foi proposto um novo algoritmo genético (NGA) para resolver FJSP de forma a minimizar *makespan*, onde uma nova representação cromossômica foi utilizada e adotou-se diferentes estratégias para efetuar cruzamento e mutação. O algoritmo proposto foi validado em uma série de conjuntos de dados de referência e também testados com dados de uma fabricante de medicamentos. Assim, NGA se mostrou mais eficiente e competitivo do que alguns outros algoritmos já existentes.

Em artigo que também trata da utilização de técnicas de GA em problemas de FJSP, Huang e Yang (2019) investiga o MOFJSP (*multi-objective flexible job-shop scheduling problem*) considerando o tempo de transporte, isto é, considerando o tempo que ocorre entre as operações agendadas nas máquinas. Nele, uma abordagem com algoritmo genético (GA) híbrido é integrado com recozimento simulado para resolver o MOFJSP

considerando o tempo de transporte, quando esse tempo não é insignificante. Uma biblioteca externa de memória de elitismo é empregado como uma biblioteca de conhecimento para direcionar a busca do GA para a região de melhor desempenho. Como resultado, este algoritmo mostrou desempenho melhor do que o GA original em termos de qualidade e distribuição de soluções, em especial com relação à flexibilidade no aumento de máquinas.

Outra técnica para resolver problemas de FJSP é a *particle swarm optimization* (PSO). Teekeng Arit Thammano (2016) propôs um novo algoritmo, denominado EPSO, para resolver FJSP baseado em PSO (*Particle Swarm Optimization*). No EPSO inclui dois conjuntos de recursos para expandir o espaço de soluções do FJSP e evitar uma convergência local ótima de forma prematura. Esses dois conjuntos são os seguintes:

I. Ciclo de vida da partícula, que consiste em quatro características: (1) cortejo de chamada, aumentando o número da prole mais eficaz (novas soluções); (2) estimulação de postura, aumentando o número de descendentes dos melhores pais (soluções atuais); (3) reprodução biparental, aumentando a diversidade da próxima geração (iteração) de soluções; (4) rotatividade da população, sucedendo a população (conjunto atual de todas as soluções) da geração anterior, por uma população (nova geração) que seja tão capaz, mas mais diversa que o anterior.

II. Consiste de mecanismo de atualização de posição discreta, movendo partículas (soluções) em direção ao líder de voo (melhor solução), ou seja, trocando alguns inteiros em cada solução com aqueles da melhor solução, usando estratégia de enxame semelhante ao procedimento de atualização do PSO contínuo. A função objetivo básica usada foi minimizar o *makespan*, que é o objetivo mais importante, portanto, fornecendo a maneira mais simples de medir a eficácia das soluções geradas. Ao se comparar o EPSO, utilizando 20 instâncias de *benchmark* bem conhecidas, contra dois métodos de otimização amplamente divulgados, demonstrou-se que o EPSO obteve um desempenho tão bom ou melhor do que os outros dois.

Em Perez-Rodriguez (2018), foi proposto uma abordagem de *Pareto*, baseada na hibridização de um EDA e *Mallows Distribution*, de forma a se construir melhores sequências para os problemas FJSP, além de resolver objetivos conflitantes. Esta abordagem híbrida explora a informação do fronte de *Pareto* usada como um parâmetro de entrada na *Mallows Distribution*. Várias instâncias e experimentos numéricos são apresentados para ilustrar que o desempenho no chão de fábrica pode ser visivelmente melhorado usando a abordagem proposta. Além disso, testes estatísticos foram executados para validar esta pesquisa.

Em Carvalho (2015) foi desenvolvido um algoritmo para tratar o FJSP com múltiplos objetivos, baseado em técnicas de computação evolutiva com objetivo de alcançar e

melhorar soluções já existentes. Onde o uso dessas técnicas se deu devido a capacidade de manusear grande número de soluções candidatas, o que faz delas apropriadas aos problemas FJSP. O algoritmo proposto fez uso de algoritmos híbridos com intuito de contornar aspectos relativos à convergência rápida que ocorre na aplicação em certas técnicas, ao mesmo tempo que mantém a diversidade favorecida por outras. Utilizou-se então um algoritmo baseado em PSO (*Particle Swarm Optimization*) associado a operadores genéticos, assim evitando a convergência prematura, mantendo a diversidade da população e, consequentemente, explora o espaço de busca de forma eficiente apresentando boas soluções se comparado com outro trabalhos encontrados na literatura.

Em Carvalho (2021), são propostos dois indicadores, Taxa de Concentração (*Concentration Rate*, CR) e Taxa de Diversidade (*Diversity Rate*, DR), para analisar características comportamentais e determinar um ponto de parada para AEMO's (Algoritmos Evolutivos Multiobjetivo), além de um indicador que pode ser utilizado na determinação do ponto de convergência de um AEMO, Indicador de Estabilidade no Espaço de Soluções (*Stability in Solution Space*). Sendo assim, para realização dos experimentos, dois AEMO's foram propostos, o SEDA (*Simple Estimation of Distribution Algorithm*) e o SEDASI (*Simple Estimation of Distribution Algorithm with Swarm Intelligence*), para tratar o problema FJSP. Os experimentos realizados, que contemplaram os AEMO's e indicadores propostos e da literatura, foram realizados com o problema FJSP e com o *benchmark* ZDT. Os resultados demonstraram a efetividade dos indicadores em apresentar as características pretendidas, e em facilitar a identificação dos pontos de interrupção e convergência de AEMOs, alcançando resultados competitivos em comparação com métodos conhecidos na literatura.

3 Desenvolvimento

Este capítulo detalha os materiais e métodos empregados no desenvolvimento deste trabalho, além disso, é apresentada a descrição do algoritmo EDA-híbrido utilizado como base e as alterações propostas para adaptação do código existente para resolução de problemas de escalonamento FJSP. Por fim, a aplicação do algoritmo proposto é exemplificada com os dados de uma matriz KC $4x5$.

3.1 Materiais e Métodos

Foi desenvolvido um algoritmo para resolução dos problemas de escalonamento do tipo FJSP, baseado na implementação e aprimoramento de um algoritmo desenvolvido pelo aluno Rocha (2023) do Programa de Mestrado e Doutorado em Computação da UFU, que foi criado inicialmente para resolver problemas de escalonamento HCSP.

Tanto HCSP quanto FJSP são classificados como problemas de escalonamento. Assim, foram realizados ajustes nos parâmetros de entrada, alterações de funções existentes e criação de funções específicas inerentes ao novo problema.

Foram utilizados conjuntos de dados amplamente conhecidos, como Kacem e Brandimarte, com a finalidade de avaliar a performance do algoritmo. Na etapa final, foram analisados os resultados obtidos e comparados a trabalhos que utilizam outras metaheurísticas, a fim de verificar se os problemas aos quais foram aplicados o algoritmo, apresentaram resultados satisfatórios.

O tópico a seguir introduz o algoritmo utilizado como base para a elaboração deste trabalho e apresenta as modificações realizadas.

3.2 Descrição do Algoritmo EDA-Híbrido

Devido ao fato desses problemas de escalonamento não possuírem algoritmos exatos que possam obter soluções ótimas, há a necessidade de aplicação de técnicas não convencionais para obtenção de resultados satisfatórios, muitas vezes com grandes quantidades de tarefas a serem distribuídas, sendo assim, utilizou-se de operadores genéticos, dentre eles, mutação, cruzamento e elitismo, os quais são aplicados sobre uma quantidade de indivíduos gerados aleatoriamente.

Cada operador genético utilizado no programa foi estruturado como uma função individual, formando assim uma biblioteca de funções, utilizadas dentro de laços, sempre

que necessário, e atuando sobre os indivíduos que haviam sido gerados inicialmente e nas gerações seguintes.

O funcionamento do algoritmo ocorre da seguinte maneira: o algoritmo inicia a partir da geração de uma população aleatória de indivíduos. A partir disso, o fluxo de execução adentra um laço de repetição que é controlado pela quantidade de gerações, especificado como parâmetro do experimento. Dentro, é realizado a criação de uma nova população com a aplicação do EDA, em conjunto com uma matriz probabilística e operadores evolutivos selecionados. Por fim, os indivíduos gerados são avaliados e ranqueados por meio de uma função de aptidão, neste caso o *fitness*, com os melhores indivíduos sendo usados na criação da próxima população. O algoritmo 2 ilustra esse processo.

Algoritmo 2 Algoritmo Base Adaptado para FJSP

1: $g \leftarrow 0$
2: Gerar população inicial
3: **enquanto** Critério de parada ou aceite não satisfeito **faça**
4: Gerar nova população com EDA + Matriz Probabilística + operadores evolutivos selecionados
5: Avaliar indivíduos com base na função de aptidão
6: Selecionar melhores indivíduos para formar a nova população
7: **fim enquanto**

3.2.1 Implementação do algoritmo

Os códigos utilizados e desenvolvidos neste trabalho utilizam a linguagem de programação Python, na versão 3.7. A geração da matriz de probabilidades, operadores evolutivos e outras rotinas FJSP utilizam cálculos implementados pela biblioteca Numpy.

O algoritmo foi implementado em um ambiente de linha de comando, recebendo uma série de parâmetros de entrada, como valores para variáveis dos operadores genéticos e funções do modelo probabilístico.

No algoritmo, um indivíduo é representado por um vetor no qual todas as operações de todos os *jobs* são ordenadas sequencialmente. As operações do primeiro *job* são dispostas em ordem, seguidas pelas operações do próximo *job*, e assim por diante. Os valores de cada posição do vetor correspondem às máquinas em que serão executadas as respectivas tarefas aos quais cada posição representa.

Ao calcular a soma dos tempos de todas as tarefas presentes em todos os *jobs* de um indivíduo, obtemos o *fitness*, que, em resumo, representa o tempo total de execução do indivíduo.

A *tarefa* é a unidade atômica de carga de trabalho que demanda certo tempo para ser executada em uma determinada unidade ou entidade computacional, sendo o ente a ser escalonado e atribuída a uma máquina. Como se trata de um escalonamento

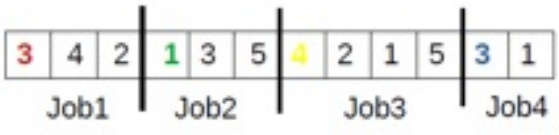

Figura 4 – Indivíduo do FJSP. Fonte: Autoria Própria.

não preemptivo, não pode ser dividida em frações, nem interrompida após ocorrer sua atribuição a uma máquina. Por sua vez, os *jobs* representam um conjunto de tarefas a serem escalonadas.

Já o *fitness* é a medida que quantifica o quão bom é um indivíduo gerado, sendo o fitness uma atribuição de valor numérico à solução (indivíduo encontrado), com base nos critérios e objetivos buscados no problema em questão. Neste caso, o fitness dos indivíduos é o tempo de conclusão das tarefas na sequência representada pelo indivíduo. Como exemplo, na Figura 5 o valor do fitness é 25, isto é, o tempo gasto para executar todas as tarefas/operações do indivíduo da Figura 4.

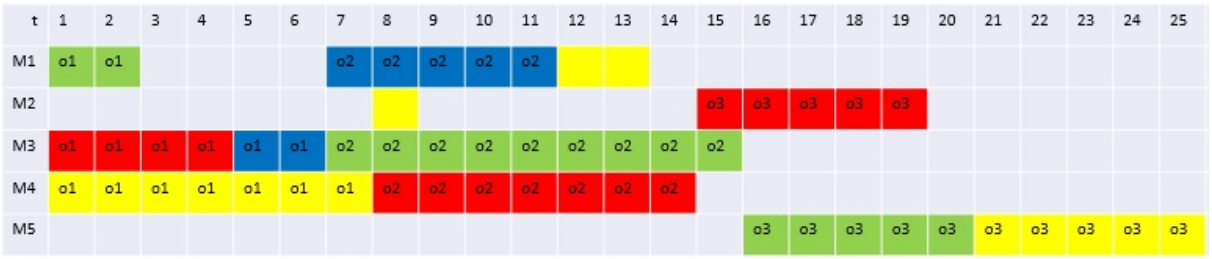

Figura 5 – Fitness. Fonte: Autoria Própria.

Por fim, *Makespan* é o critério utilizado para comparar os resultados da heurística, e representa o menor valor máximo das execuções. Em outras palavras, o valor mínimo possível e necessário para executar todas as tarefas/operações para um indivíduo. Na Figura 6 tem-se o valor de 11 unidades de tempo, onde todas as tarefas foram executadas de forma otimizada.

O programa recebe, por linha de comando os parâmetros de entrada, como valores das variáveis referentes aos operadores genéticos, executa as funções relativas ao modelo probabilístico e aos operadores genéticos, e ao fim da execução retorna os valores de makespan.

Um dos parâmetros fornecidos na linha de comando é o caminho do diretório que contém o arquivo com as informações da matriz ETC (figura 7). Nesta matriz, as operações de cada *job* são representados pelas linhas, ficando os *jobs* de forma sequencial linha a linha, sendo as máquinas representadas pelas colunas e cada célula da matriz indica o tempo demandado por cada tarefa em diferentes máquinas disponíveis, isto é, qual o

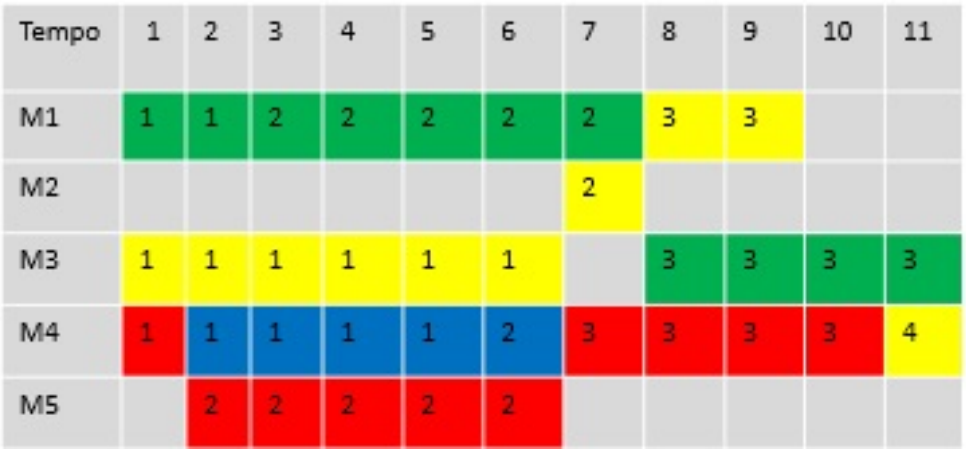

Figura 6 – Makespan. Fonte: Autoria Própria.

tempo demandado por cada tarefa em cada máquina disponível. Para cada instância de problema FJSP, proposta por Kacem, Hammadi e Borne (2002), foi criado um arquivo com a ETC correspondente.

Job	M1	M2	M3	M4	M5
J1	2	5	4	1	2
	5	4	5	7	5
	4	5	5	4	5
J2	2	5	4	7	8
	5	6	9	8	5
	4	5	4	54	5
J3	9	8	6	7	9
	6	1	2	5	4
	2	5	4	2	4
	4	5	2	1	5
J4	1	5	2	4	12
	5	1	2	1	2

Figura 7 – Tabela ETC para FJSP. Fonte: Kacem, Hammadi e Borne (2002).

3.2.2 Funções do algoritmo

Nesta subsessão, são apresentadas as principais funcionalidades do código inicial, que serviu como base para este trabalho. Além disso, são mostrados os códigos desenvolvidos para as novas funcionalidades envolvidas na resolução do problema de escalonamento FJSP.

3.2.2.1 Geração da população inicial

A rotina que realiza a geração da população inicial aleatória de cada experimento recebe o número de indivíduos desejados na população, número máximo de máquinas disponíveis e número de tarefas a serem atribuídas, e constrói uma matriz com nxm, onde n corresponde ao número de indivíduos e m representa a quantidade de tarefas, com cada célula contendo um valor aleatório entre 0 e o número de máquinas.

3.2.2.2 Ordenar população

A função Ordenar população, recebe como entrada a matriz de tempos de execução, uma população de indivíduos e uma sequência de tarefas a serem escalonadas. Ela calcula o *makespan* de cada indivíduo na população, utilizando de processamento paralelo para melhorar sua eficiência. Em seguida, ordena os indivíduos com base nos *makespans* calculados, retornando a população ordenada. Isso garante que os indivíduos com tempos de execução melhores sejam priorizados.

3.2.2.3 Modelo probabilístico e Matriz de probabilidade

A rotina que realiza a geração da matriz de probabilidade implementa o modelo probabilístico. Tal modelo representa um papel central no EDA e é baseado no paradigma do modelo gráfico probabilístico (LARRAÑAGA, 2002).

O modelo probabilístico é o responsável por gerar novos indivíduos, ou seja, direcionar o algoritmo no espaço de busca. Isto é, ela objetiva dar suporte na formação de novos indivíduos, de modo que as escolhas de cada elemento do indivíduo sejam feitas de forma probabilística, utilizando as informações do modelo (CARVALHO, 2015).

Assim, sua estrutura e construção estão associadas à definição do problema e à representação da solução. Se um indivíduo (Figura 8) é formado por um vetor de m posições e cada posição possui n possibilidades de valor, uma possível proposta para modelo probabilístico é uma matriz nxm, em que a posição ixj representaria a probabilidade da posição j do vetor assumir o valor representado pela linha i, na respectiva coluna j. Com isso, o modelo probabilístico, utiliza uma matriz, denominada por matriz de probabilidades, onde cada coluna dessa matriz contém a quantidade de vezes que aquela operação aparece em cada máquina (ROCHA, 2023).

A Matriz de Probabilidade (MP), representada pela Figura 9, é uma matriz de tamanho nxm, onde cada linha i, representa as operações e cada coluna j, representa uma máquina. O percentual dos melhores indivíduos utilizados para sua formação corresponde a um total de N indivíduos, e ela é formada analisando esses N indivíduos e contando o número de aparições de cada associação entre uma máquina e uma tarefa. Portanto,

selected machine	2	1	2	3	1	0
task number	0	1	2	3	4	5

selected machine	3	2	0	1	0	2
task number	0	1	2	3	4	5

selected machine	0	3	1	2	3	1
task number	0	1	2	3	4	5

Figura 8 – Indivíduo Hipotético. Fonte: Rocha (2023).

		Máquinas			
		0	**1**	**2**	**3**
Tarefas	**0**	1	0	1	1
	1	0	1	1	1
	2	1	1	1	0
	3	0	1	1	1
	4	1	1	0	1
	5	1	1	1	0

Figura 9 – Matriz de Probabilidade. Fonte Rocha (2023).

cada posição i, j de MP contém o número de associações entre i e j que ocorreram nos indivíduos N.

3.2.2.4 Preenchimento da matriz de probabilidade

O procedimento que realiza o preenchimento da matriz de probabilidade recebe três parâmetros de entrada, a matriz de probabilidades a ser preenchida, a população de indivíduos e número de tarefas do experimento. A atribuição de valores acontece por meio de um laço de repetição, que é controlado pela quantidade de indivíduos, e onde é realizado a contagem da frequência em que as tarefas foram atribuídas a determinadas máquinas dentro da população, que é atribuída a cada célula da matriz, que é retornada ao final do processo. Essa função permite que a matriz capture a distribuição das atribuições de tarefas às máquinas nos diversos indivíduos da população.

3.2.2.5 Função roleta

A função roleta realiza uma seleção probabilística na população de indivíduos. Ela calcula pesos ponderados para cada indivíduo e os normaliza para garantir que a soma seja 1. Em seguida, cria uma distribuição de probabilidades com base nos pesos. Um número aleatório é gerado e é encontrada a posição correspondente na distribuição cumulativa de probabilidades. Isso resulta na seleção de um índice na população com base nas probabilidades dos pesos. Esse índice selecionado é retornado como o indivíduo escolhido para reprodução.

3.2.2.6 Geração de novos indivíduo

A função criar indivíduo cria um novo indivíduo com base em uma matriz de probabilidade e função roleta. Um novo indivíduo é gradualmente construído iterando sobre as linhas da matriz de probabilidade. Para cada linha, um índice é selecionado com base nas probabilidades definidas na mesma. Esse índice é então adicionado ao novo indivíduo, que é gradualmente construído ao longo das iterações. O novo indivíduo é retornado no final da função.

3.2.2.7 Gerar nova população

A função gerar nova população gera uma nova população de indivíduos com base em uma matriz de probabilidade, na matriz de tempos e na função roleta. Ela itera sobre o número de indivíduos desejados, chamando a função de criação de novos indivíduos para criar cada indivíduo. Os indivíduos são adicionados a uma lista e, no final, essa lista é retornada, sendo esta uma nova população com valores aprimorados de *fitness*.

3.2.2.8 Mutação

Tendo em mente que a mutação é um operador genético que busca introduzir pequenas alterações aleatórias nos indivíduos de forma a explorar novas soluções no espaço de busca, nesta função, busca-se introduzir aleatoriedade tanto na seleção do indivíduo, quanto na posição onde essa mutação ocorreria nos indivíduos selecionados a sofrer mutação.

Assim, a função recebe quatro argumentos como entrada. O primeiro argumento corresponde a uma lista de indivíduos que compõem uma população. O segundo argumento é a probabilidade de mutação, e seu valor é um número inteiro entre 1 e 100. O terceiro corresponde ao número de máquinas disponíveis, e o quarto é a matriz de tempos demandados para execução de cada tarefa em cada uma das máquinas disponíveis.

Inicia-se, assim, um laço que irá iterar sobre cada indivíduo na população. Para introduzir aleatoriedade na função, um número inteiro aleatório é gerado entre 1 e 100,

representando uma probabilidade de mutação. Em seguida, verifica-se se esse número está dentro da probabilidade de mutação especificada. Se estiver, a mutação é aplicada ao indivíduo; caso contrário, nenhum ajuste é realizado.

Para cada indivíduo selecionado para mutação, é escolhida aleatoriamente uma posição na sua sequência de tarefas. Em seguida, outra posição é escolhida aleatoriamente. As tarefas nessas posições são trocadas entre si.

O laço continuará a iterar sobre todos os indivíduos da população, sendo a mutação aplicada àqueles indivíduos que atenderam à condição de probabilidade.

3.3 Alterações para Resolução de Problemas FJSP

Para adaptar o código inicial para a resolução de problemas de escalonamento FJSP, foi necessário a criação de um novo parâmetro de entrada e diversas rotinas para viabilizar a implementação do novo procedimento de cálculo de *fitness*.

A primeira alteração realizada consistiu na adição de um novo argumento de linha de comando *seq_jobs*, uma lista de números inteiros correspondente ao número de operações que compõe cada *job* dentro do experimento. Sendo necessário inseri-lo de maneira sequencial, por exemplo, um valor $3, 4, 2$, indica que a primeira *job* é composta por três operações, enquanto as *jobs* seguintes contem quatro e duas operações, respectivamente.

Esse novo parâmetro foi acrescentado em preparação para a codificação da nova função de cálculo de *fitness*, sendo utilizada para criar partições distintas no vetor de operações, de maneira sequencial, contendo a respectiva máquina onde cada tarefa da *job* será executada. A solução proposta para o cálculo de *makespans* é apresentada na subseção 3.3.3, enquanto as rotinas auxiliares são descritas nas subseções 3.3.1 e 3.3.2.

3.3.1 Recuperar sub jobs

No contexto de resolução de problemas de escalonamento FJSP, as *jobs* são decompostas em operações individuais, com cada uma sendo atribuída a uma respectiva máquina para execução. A função que realiza essa divisão é ilustrada no Algoritmo 3, que recebe a população de indivíduos (individuos) e a lista contendo a quantidade de operações de cada *job* em sequência (seq_jobs).

Nas linhas 2, 3 são realizadas instanciações de dois valores, uma lista que armazenará em qual máquina cada tarefa da *job* será executada e uma lista que guarda o índice inicial de cada partição dentro do vetor de operações, respectivamente. Uma variável contadora é inicializada na linha 4, com o objetivo de auxiliar nesta tarefa.

Após a inicialização, a rotina adentra um laço de repetição, entre as linhas 5 e 9, que itera sobre os elementos da lista seq_jobs. Nas linhas subsequentes, as máquinas

Algoritmo 3 Rotina para Recuperação de Sub Jobs

```
1:  função RecuperarSubJobs(individuos, seq_jobs)
2:      sub_jobs ← ListaVazia()
3:      start_indexes ← ListaVazia()
4:      start_index ← 0
5:      para i em seq_jobs faça
6:          Adicione máquinas correspondentes à job i em sub_jobs
7:          Adicione start_index à lista start_jobs
8:          start_index ← start_index + i
9:      fim para
10:     Retorne sub_jobs, start_indexes
11: fim função
```

correspondentes à *job* são extraídas do vetor de indivíduos. Os índices que devem ser extraídos são obtidos a partir da soma da variável contadora *start_index* com a quantidade de operações i. Por fim, o índice inicial é acrescentado à lista *start_indexes* e incrementado. Ao fim desse processo, é retornado uma tupla com dois elementos, contendo as partições de máquinas e índices de início correspondentes a cada *job*.

3.3.2 Recuperar máquinas

Para possibilitar a execução sequencial, característica de problemas de escalonamento FJSP, foi criada a rotina Recuperar Maquinas, representada no Algoritmo 4. Essa função foi criada com o objetivo de recuperar as máquinas onde serão executadas a n-ésima operação de todos os *jobs* do experimento e deve ser invocada com o vetor de máquinas particionado em *jobs* (sub_jobs) e um valor inteiro que informa o índice da n-ésima operação que será executada (job_number).

A linha 2 inicializa uma lista de inteiros vazia, com o objetivo de guardar as máquinas onde a n-ésima operação de todas as *jobs* serão armazenadas. Esse vetor será preenchido no laço de repetição que abrange as linhas entre 3 e 9. Internamente, é realizado uma iteração em todas as partições e verificado, na linha 4, se a respectiva divisão possui uma operação na n-ésima posição e resgata a máquina onde ela será executada. Caso contrário, é realizado o preenchimento da lista de máquinas com o valor constante -1, com o propósito exclusivo de simplificar o processamento realizado no cálculo de *makespan*. Ao término de todas as iterações do laço, a lista de máquinas é retornada.

3.3.3 Calcular fitness

A função Calcular Fitness, descrita no Algoritmo 5, desempenha um papel essencial neste trabalho, sendo responsável pelo cálculo de tempo de execução de todas as máquinas, considerando as operações realizadas e os indivíduos gerados previamente, e em conformidade às restrições estabelecidas por problemas FJSP. Essa rotina recebe uma

Algoritmo 4 Função para recuperar máquinas da n-ésima operação

```
1: função RecuperarMaquinas(jobs, job_number)
2:     maquinas ← ListaVazia()
3:     para particao em jobs faça
4:         se Existe n-ésima operação na partição então
5:             Adicione máquina correspondente à operação n em maquinas
6:         senão
7:             Adicione -1 em maquinas
8:         fim se
9:     fim para
10:    Retorne maquinas
11: fim função
```

matriz ETC, uma população de indivíduos (individuos) e a lista contendo a quantidade de operações de cada *job* em sequência (seq_jobs). A modelagem proposta para o algoritmo envolve escalonar as operações por ordem de sequência das *jobs*, dessa forma, em cada iteração de um laço de repetição externo, o tempo de execução da i-ésima operação é somado na sua respectiva máquina.

Algoritmo 5 Função para cálculo de *fitness*

```
1: função CalcularFitness(ET, individuos, seq_jobs)
2:     qtd_operacoes ← Maximo(seq_jobs)
3:     jobs, indices ← RecuperarSubJobs(individuos, seq_jobs)
4:     makespans ← VetorNumpy(QtdMaquinas)
5:     tempos ← FilaVazia()
6:     para i de 0 até qtd_operacoes faça
7:         maquinas ← RecuperarMaquinas(jobs, i)
8:         para maquina em maquinas faça
9:             se houver operação para ser executada então
10:                tempo ← 0
11:                se não for a primeira operação então
12:                    ultimo_tempo ← Desenfileirar(tempos)
13:                    tempo ← Maximo(ultimo_tempo, makespans[maquina])
14:                    tempo ← tempo + makespans[maquina] + ET[tarefa][maquina]
15:                senão
16:                    tempo ← makespans[maquina] + ET[tarefa][maquina]
17:                fim se
18:                makespans[maquina] ← tempo
19:                se job possui outra operação então
20:                    tempos ← tempos.Enfileirar(tempo)
21:                fim se
22:            fim se
23:        fim para
24:    fim para
25:    retorna Maximo(makespans)
26: fim função
```

No contexto de problemas de escalonamento FJSP, cada *job* pode ser executada de maneira paralela, embora cada máquina não seja capaz de executar mais de uma operação no mesmo tempo t. Essa restrição é modelada com a introdução de uma estrutura de fila, escolhida por apresentar performance superior quando as operações ocorrem em suas extremidades, onde o tempo de término da última operação da respectiva *job* é enfileirado, sendo posteriormente comparado ao tempo da última atividade na máquina-alvo, com o maior valor selecionado e somado ao tempo da máquina correspondente. Caso a última operação da respectiva *job* executada em uma máquina n seja finalizada após o tempo de término da última atividade em uma máquina-alvo m, a diferença entre esses valores corresponde ao tempo em que a máquina-alvo permaneceu ociosa, aguardando a atribuição de uma tarefa.

Sendo assim, a linha 2 instancia uma variável de controle, preenchida com a quantidade máxima de operações dentro de uma *job* do respectivo experimento. Em seguida, a linha 3 realiza uma chamada ao procedimento ”Recuperar Sub Jobs”, descrito anteriormente na subseção 3.3.1, e atribui seu resultado a duas variáveis distintas, contendo as máquinas que executarão as operações particionadas por *job*, e seus respectivos índices de início nas linhas da matriz ETC. Por fim, as linhas 4 e 5 realizam a inicialização de um vetor Numpy que armazena o tempo de término de execução de cada máquina e uma fila que expressa as restrições decorrentes da execução paralela do FJSP, respectivamente.

A execução sequencial é expressa pelo laço de repetição iniciado na linha 6, garantindo que sejam somados somente os tempos da n-ésima operação de cada *job*. Em sequência, é realizado uma chamada à rotina ”Recuperar Maquinas”, descrita na subseção 3.3.2, retornando uma lista contendo a máquina de execução da operação n ou o valor constante ”-1” para indicar ausência de tarefas, sendo utilizada posteriormente na linha 8 para iteração. Por sua vez, o índice da tarefa correspondente na matriz ETC é extraído a partir do índice inicial da *job* no vetor de operações somado ao índice i.

Os trechos de código presentes entre as linhas 9 e 22 serão executadas caso haja uma máquina para executar a n-ésima operação da *job*, em termos práticos, caso o índice da máquina seja diferente da constante ”-1”. O tempo de execução das operações são somados ao vetor de *makespans*, e enfileiradas, caso a respectiva *job* possua outra operação. A diferença ocorre quando as primeiras operações de todas as *jobs* já foram executadas, com o fluxo de execução adentrando o código interno do ramo condicional da linha 11, responsável por expressar as restrições de execução paralela do FJSP. A partir daqui, o tempo considerado para incremento no vetor de *makespans* é dado pelo maior valor entre o tempo de término da última operação da respectiva *job* e tempo de término da última operação na máquina-alvo, somado ao tempo de execução, para a respectiva tarefa, extraído da matriz ETC.

Por fim, o valor de *makespan* para a população de indivíduos é dado pelo valor

máximo do vetor de *makespans* e retornado para análise em outras rotinas pré-existentes.

A Figura 10 apresenta um exemplo da execução e distribuição do tempo gasto por todas operações que compõe um indivíduo, de acordo com a atribuição de máquinas e uma tabela de tempos (ETC). Na Figura 10 (A), temos a representação de um indivíduo gerado aleatoriamente. Na Figura 10 (B), temos uma Matriz de Tempos (ETC), que neste caso utilizou-se o problema $4x5$ proposto por kacem. Na Figura 10 (C), uma simulação do cálculo do *fitness*, com a distribuição dos tempos de execução de cada operação preenchida ao longo do eixo dos tempos, isto é, ao longo das linhas, com o tempo indicado pelas colunas, enquanto cada uma das linhas correspondentes às máquinas disponíveis para execução das operação.

No exemplo da Figura 10 (A), o indivíduo é composto por 4 *Jobs*, que por sua vez possuem diferentes quantidade de operações. Por exemplo, o *Job1* é composto por 3 operações, o Job2 é composto por 3 operações, o *Job3* é composto por 4 operações enquanto o *Job4* é composto por 2 operações.

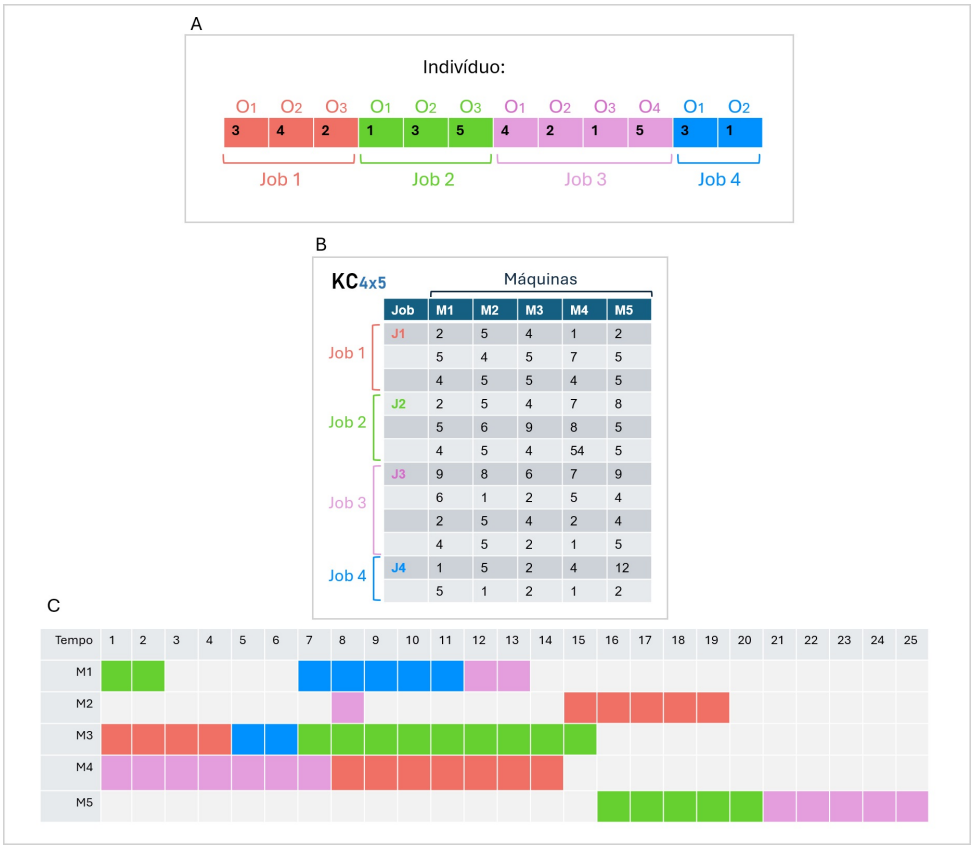

Figura 10 – Execução. Fonte: Autoria Própria.

Os valores das células do vetor que representa o indivíduo correspondem às máquinas as quais as operações serão executadas. Neste caso, a primeira posição corresponde a primeira operação do *Job1* e está atribuída à máquina 3. Enquanto a segunda operação, que corresponde a posição 2 do vetor, indica que ela foi atribuída à máquina 4. E assim sucessivamente.

Para obtenção do tempo de execução dessas operação, deve-se consultar a tabela ETC demonstrada na Figura 10 (B). Cada linha dessa tabela indica os tempos de cada uma das operações do indivíduo para todas as máquinas disponíveis, indicadas pelas Colunas. Como exemplo, a primeira operação do *Job1*, se atribuída à máquina 3, conforme 10 (A), utilizará 4 unidades de tempo, conforme indicado na Primeira linha e coluna M3 da Figura 10 (B). Enquanto a operação2 do *Job1*, atribuída a máquina 4, necessita de 7 unidade de tempo. Esse processo de obtenção dos tempos se repete para todas as demais operações.

As operações de cada *job* devem ser executadas de forma sequencial, isto é, a operação 2 de um *job* só pode começar quando a operação 1 desse mesmo *job* finalizar. Como exemplo temos as operações do *Job4*, onde a segunda operação começa apenas no tempo 7, já que a operação 1 finalizou no tempo 6.

Além disso, como cada máquina só poderá executar apenas uma operação por vez, o tempo de execução do próximo *Job* nesta máquina deve ocorrer após ela finalizar. Como acontece com a operação 1 do *Job4*, que começa apenas no tempo 5, já que antes disso a máquina está ocupada executando a operação 1 do *Job1*.

A sequencia da soma dos tempos de cada máquina, e preenchimento da Figura 10 (C), começa sempre pelas primeiras operações de cada *job*, após obtenção e preenchimento de todas as primeiras operações, passa-se para as segundas operações e assim sucessivamente.

Ao final, após todos os tempos de todas as operações terem sido somados aos tempos das máquinas as quais eles foram atribuídos, verifica-se qual a máquina possui o maior tempo acumulado. Este valor será o *fitness* do indivíduo.

Esse processo se repete para todos os demais indivíduos gerados, e o maior fitness encontrado dentre todos os indivíduos, ser o valor de makespan.

4 Resultados e Discussão

Como parâmetros de entrada, são informadas a quantidade de indivíduos a serem gerados de forma aleatória, a porcentagem de mutação a ser aplicada para a próxima geração, a quantidade de gerações, a porcentagem de cruzamentos, além do porcentual de elitismo a ser considerado e o caminho da ETC, que é a tabela de tempos estimados que cada tarefa demanda para ser executada em cada uma das máquinas possíveis. A execução termina após rodar todos as funções e laços usando os parâmetros de entrada. Para realizar múltiplas execuções com valores e parâmetros diferentes, há a possibilidade de adicionar múltiplos valores para cada parâmetro, assim o algoritmo roda de forma sequencial cada um dos valores definidos de um parâmetro específico.

4.1 Experimentos

A coleta de dados foi realizada utilizando os 4 problemas propostos por Kacem, Hammadi e Borne (2002) e que estão especificados nas tabelas 1, 2, 3, 4 e 5. Como o objetivo é obter um *makespan* cada vez menor, o que significa encontrar um tempo menor para execução de todas as operações de todos os *jobs*, e como a variação de cada parâmetro impacta de alguma forma no valor obtido, ao fazermos diversas composições entre os valores de parâmetros em cada execução, obtemos diferentes valores de *makespan* que variam de acordo com a variação nos parâmetros.

Assim, com base nos resultados obtidos nas modificações, pôde-se selecionar os melhores valores de cada parâmetro para que fossem testados os demais.

Para cada combinação de valores de variáveis, foram feitas 30 execuções a fim de verificar o desvio nos valores de resultado.

Os valores utilizados foram os seguintes:

- Mutação: 5%.
- Elitismo: 20% e 30%.
- Número de indivíduos: 50, 100, 500 e 2000.
- Número de gerações: 50 e 100

Os valores acima citados foram os que apresentaram os melhores resultados, sendo que valores acima dos maiores utilizados ou menores do que os mínimos considerados não demonstraram desempenho satisfatório.

Tabela 1 – Problema 4 x 5 de Kacem, Hammadi e Borne (2002)

Job	$\mathbf{O}_i$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$
J_1	O_1	2	5	4	1	2
	O_2	5	4	5	7	5
	O_3	4	5	5	4	5
J_2	O_1	2	5	4	7	8
	O_2	5	6	9	8	5
	O_3	4	5	4	54	5
J_3	O_1	9	8	6	7	9
	O_2	6	1	2	5	4
	O_3	2	5	4	2	4
	O_4	4	5	2	1	5
J_4	O_1	1	5	2	4	12
	O_2	5	1	2	1	2

Tabela 2 – Problema 8 x 8 de Kacem, Hammadi e Borne (2002)

$\mathbf{J}_i$	$\mathbf{O}_i$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$	$\mathbf{M}_8$
J_1	O_1	5	3	5	3	3	-	10	9
	O_2	10	-	5	8	3	9	9	6
	O_3	-	10	-	5	6	2	4	5
J_2	O_1	5	7	3	9	8	-	9	-
	O_2	-	8	5	2	6	7	10	9
	O_3	-	10	-	5	6	4	1	7
	O_4	10	8	9	6	4	7	-	-
J_3	O_1	10	-	-	7	6	5	2	4
	O_2	-	10	6	4	8	9	10	-
	O_3	1	4	5	6	-	10	-	7
J_4	O_1	3	1	6	5	9	7	8	4
	O_2	12	11	7	8	10	5	6	9
	O_3	4	6	2	10	3	9	5	7
J_5	O_1	3	6	7	8	9	-	10	-
	O_2	10	-	7	4	9	8	6	-
	O_3	-	9	8	7	4	2	7	-
	O_4	11	9	-	6	7	5	3	6
J_6	O_1	6	7	1	4	6	9	-	10
	O_2	10	-	9	9	9	7	6	4
	O_3	11	5	9	10	11	-	10	-
J_7	O_1	5	4	2	6	7	-	10	-
	O_2	-	9	-	9	11	9	10	5
	O_3	-	8	9	3	8	6	-	10
J_8	O_1	2	8	5	9	-	4	-	10
	O_2	7	4	7	8	9	-	10	-
	O_3	9	9	-	8	5	6	7	1
	O_4	9	-	3	7	1	5	8	-

Tabela 3 – Problema 10 x 7 de Kacem, Hammadi e Borne (2002)

$\mathbf{J}_i$	$\mathbf{O}_i$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$
J_1	O_1	1	4	6	9	3	5	2
	O_2	8	9	5	4	1	1	3
	O_3	4	8	10	4	11	4	3
J_2	O_1	6	9	8	6	5	10	3
	O_2	2	10	4	5	9	8	4
J_3	O_1	15	4	8	4	8	7	1
	O_2	9	6	1	10	7	1	6
	O_3	11	2	7	5	2	3	14
J_4	O_1	2	8	5	8	9	4	3
	O_2	5	3	8	1	9	3	6
	O_3	1	2	6	4	1	7	2
J_5	O_1	7	1	8	5	4	3	9
	O_2	2	4	5	10	6	4	9
	O_3	5	1	7	1	6	6	2
J_6	O_1	8	7	4	56	9	8	4
	O_2	5	14	1	9	6	5	8
	O_3	3	5	2	5	4	5	7
J_7	O_1	5	6	3	6	5	15	2
	O_2	6	5	4	9	5	4	3
	O_3	9	8	2	8	6	1	7
J_8	O_1	6	1	4	1	10	4	3
	O_2	11	13	9	8	9	10	8
	O_3	4	2	7	8	3	10	7
J_9	O_1	12	5	4	5	4	5	5
	O_2	4	2	15	99	4	7	3
	O_3	9	5	11	2	5	4	2
J_{10}	O_1	9	4	13	10	7	6	8
	O_2	4	3	25	3	8	1	2
	O_3	1	2	6	11	13	3	5

Tabela 4 – Problema 10 x 10 de Kacem, Hammadi e Borne (2002)

$\mathbf{J}_i$	$\mathbf{O}_i$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$	$\mathbf{M}_8$	$\mathbf{M}_9$	$\mathbf{M}_{10}$
J_1	O_1	1	4	6	9	3	5	2	8	9	5
	O_2	4	1	1	3	4	8	10	4	11	4
	O_3	3	2	5	1	5	6	9	5	10	3
J_2	O_1	2	10	4	5	9	8	4	15	8	4
	O_2	4	8	7	1	9	6	1	10	7	1
	O_3	6	11	2	7	5	3	5	14	9	2
J_3	O_1	8	5	8	9	4	3	5	3	8	1
	O_2	9	3	6	1	2	6	4	1	7	2
	O_3	7	1	8	5	4	9	1	2	3	4
J_4	O_1	5	10	6	4	9	5	1	7	1	6
	O_2	4	2	3	8	7	4	6	9	8	4
	O_3	7	3	12	1	6	5	8	3	5	2
J_5	O_1	7	10	4	5	6	3	5	15	2	6
	O_2	5	6	3	9	8	2	8	6	1	7
	O_3	6	1	4	1	10	4	3	11	13	9
J_6	O_1	8	9	10	8	4	2	7	8	3	10
	O_2	7	3	12	5	4	3	6	9	2	15
	O_3	4	7	3	6	3	4	1	5	1	11
J_7	O_1	1	7	8	3	4	9	4	13	10	7
	O_2	3	8	1	2	3	6	11	2	13	3
	O_3	5	4	2	1	2	1	8	14	5	7
J_8	O_1	5	7	11	3	2	9	8	5	12	8
	O_2	8	3	10	7	5	13	4	6	8	4
	O_3	6	2	13	5	4	3	5	7	9	5
J_9	O_1	3	9	1	3	8	1	6	7	5	4
	O_2	4	6	2	5	7	3	1	9	6	7
	O_3	8	5	4	8	6	1	2	3	10	12
J_{10}	O_1	4	3	1	6	7	1	2	6	20	6
	O_2	3	1	8	1	9	4	1	4	17	15
	O_3	9	2	4	2	3	5	2	4	10	23

Tabela 5 – Problema 15 x 10 de Kacem, Hammadi e Borne (2002)

J	$\mathbf{O}_i$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}_4$	$\mathbf{M}_5$	$\mathbf{M}_6$	$\mathbf{M}_7$	$\mathbf{M}_8$	$\mathbf{M}_9$	$\mathbf{M}_{10}$
J_1	O_1	1	4	6	9	3	5	2	8	9	4
	O_2	1	1	3	4	8	10	4	11	4	3
	O_3	2	5	1	5	6	9	5	10	3	2
	O_4	10	4	5	9	8	4	15	8	4	4
J_2	O_1	4	8	7	1	9	6	1	10	7	1
	O_2	6	11	2	7	5	3	5	14	9	2
	O_3	8	5	8	9	4	3	5	3	8	1
	O_4	9	3	6	1	2	6	4	1	7	2
J_3	O_1	7	1	8	5	4	9	1	2	3	4
	O_2	5	10	6	4	9	5	1	7	1	6
	O_3	4	2	3	8	7	4	6	9	8	4
	O_4	7	3	12	1	6	5	8	3	5	2
J_4	O_1	6	2	5	4	1	2	3	6	5	4
	O_2	8	5	7	4	1	2	36	5	8	5
	O_3	9	6	2	4	5	1	3	6	5	2
	O_4	11	4	5	6	2	7	5	4	2	1
J_5	O_1	6	9	2	3	5	8	7	4	1	2
	O_2	5	4	6	3	5	2	28	7	4	5
	O_3	6	2	4	3	6	5	2	4	7	9
	O_4	6	5	4	2	3	2	5	4	7	5
J_6	O_1	4	1	3	2	6	9	8	5	4	2
	O_2	1	3	6	5	4	7	5	4	6	5
J_7	O_1	1	4	2	5	3	6	9	8	5	4
	O_2	2	1	4	5	2	3	5	4	2	5
J_8	O_1	2	3	6	2	5	4	1	5	8	7
	O_2	4	5	6	2	3	5	4	1	2	5
	O_3	3	5	4	2	5	49	8	5	4	5
	O_4	1	2	36	5	2	3	6	4	11	2
J_9	O_1	6	3	2	22	44	11	10	23	5	1
	O_2	2	3	2	12	15	10	12	14	18	16
	O_3	20	17	12	5	9	6	4	7	5	6
	O_4	9	8	7	4	5	8	7	4	56	2
J_{10}	O_1	5	8	7	4	56	3	2	5	4	1
	O_2	2	5	6	9	8	5	4	2	5	4
	O_3	6	3	2	5	4	7	4	5	4	1
	O_4	3	2	5	6	5	8	7	4	5	2
J_{11}	O_1	1	2	3	6	5	2	1	4	2	1
	O_2	2	3	6	5	2	1	4	10	12	1
	O_3	3	6	2	5	8	4	6	3	2	5
	O_4	4	1	45	6	2	4	1	25	2	4
J_{12}	O_1	9	8	5	6	3	6	5	2	4	2
	O_2	5	8	9	5	4	75	63	6	5	21
	O_3	12	5	4	6	3	2	5	4	2	5
	O_4	8	7	9	5	6	3	2	5	8	4
J_{13}	O_1	4	2	5	6	8	5	6	4	6	2
	O_2	3	5	4	7	5	8	6	6	3	2
	O_3	5	4	5	8	5	4	6	5	4	2
	O_4	3	2	5	6	5	4	8	5	6	4
J_{14}	O_1	2	3	5	4	6	5	4	85	4	5
	O_2	6	2	4	5	8	6	5	4	2	6
	O_3	3	25	4	8	5	6	3	2	5	4
	O_4	8	5	6	4	2	3	6	8	5	4
J_{15}	O_1	2	5	6	8	5	6	3	2	5	4
	O_2	5	6	2	5	4	2	5	3	2	5
	O_3	4	5	2	3	5	2	8	4	7	5
	O_4	6	2	11	14	2	3	6	5	4	8

Os dados gerados permitiram analisar a eficiência do algoritmo criado e comparar com resultados de outros trabalhos. Foram obtidos executando-se o algoritmo por diversas vezes. A cada execução do algoritmo, foram alterados os diversos parâmetros de entrada, como número de indivíduos a serem gerados, além de percentual de mutação, cruzamento e elitismo.

4.2 Resultados

Os resultados foram divididos em 4 tabelas, que correspondem as execuções de cada um dos problemas testados e analisados.

Em cada coluna das tabelas, temos uma combinação de valores para mutação, elitismo, Número de indivíduos, Número de gerações. Em cada linha, temos as 30 execuções, Como exemplo, na primeira coluna da tabela, temos os valores:

- Mutação: 5.
- Elitismo: 30.
- Número de indivíduos: 50.
- Número de gerações: 50.

Tabela 6 – Execuções do kacem 4x5

Problema	4x5							
Mutação	5	5	5	5	5	5	5	5
Elitismo	30	30	30	30	20	20	20	20
Indivíduos	50	50	100	100	50	50	100	100
Nº Gerações	50	100	50	100	50	100	50	100
Nº de Execuções	30	30	30	30	30	30	30	30
Menor Valor	11	11	11	11	11	11	11	11
Maior Valor	15	12	12	12	14	13	12	13
Média	11,633	11,366	11,133	11,066	11,566	11,633	11,100	11,100
Desvio Padrão	0,7951	0,4819	0,3399	0,2494	0,8439	0,6046	0,3000	0,3958

Tabela 7 – Execuções do kacem 8x8

Problema	8x8								
Mutação	5	5	5	5	5	5	5	5	5
Elitismo	30	30	30	30	20	20	20	20	20
Indivíduos	50	50	100	100	50	50	100	100	1000
Nº Gerações	50	100	50	100	50	100	50	100	100
Nº de Execuções	30	30	30	30	30	30	30	30	30
Menor Valor	24	20	19	18	23	20	17	17	15
Maior Valor	32	31	25	24	29	29	24	23	16
Média	27	25,533	21,333	21,1	25,8	25	20,533	20,833	15,866
Desvio Padrão	1,7889	2,6043	1,7951	1,7578	1,4922	2,1756	1,4996	1,3437	0,3399

Tabela 8 – Execuções do kacem 10x7

Problema	10x7								
Mutação	5	5	5	5	5	5	5	5	5
Elitismo	30	30	30	30	20	20	20	20	20
Indivíduos	50	50	100	100	50	50	100	100	1000
Nº Gerações	50	100	50	100	50	100	50	100	100
Nº de Execuções	30	30	30	30	30	30	30	30	30
Menor Valor	16	15	12	12	14	15	12	12	11
Maior Valor	21	22	16	16	20	19	17	17	12
Média	18,233	17,500	14,266	13,900	17,200	16,733	14,300	14,033	11,333
Desvio Padrão	1,4302	1,5864	1,2092	1,1060	1,5790	1,1528	1,3940	1,0796	0,4714

Tabela 9 – Execuções do kacem 10x10

Problema	10x10								
Mutação	5	5	5	5	5	5	5	5	5
Elitismo	30	30	30	30	20	20	20	20	20
Indivíduos	50	50	100	100	50	50	100	100	1000
Nº Gerações	50	100	50	100	50	100	50	100	100
Nº de Execuções	30	30	30	30	30	30	30	30	30
Menor Valor	11	10	9	8	12	11	8	9	7
Maior Valor	18	17	13	12	17	16	12	13	8
Média	13,933	13,4	10,4	9,833	13,633	13,066	10,5	10,4	7,2
Desvio Padrão	1,6111	1,5188	0,9165	1,0980	1,3287	1,2893	0,9574	1,0520	0,4000

Tabela 10 – Execuções do kacem 15x10 - Parte 1

Problema	15x10								
Mutação	5	5	5	5	5	5	5	5	5
Elitismo	30	30	30	30	20	20	20	20	20
Indivíduos	50	50	100	100	50	50	100	100	500
Nº Gerações	50	100	50	100	50	100	50	100	100
Nº de Execuções	30	30	30	30	30	30	30	30	30
Menor Valor	21	21	17	16	20	20	16	15	12
Maior Valor	28	29	21	21	27	28	21	21	14
Média	24,633	23,6	19,066	18,8	23,633	23,3	18,833	18,666	12,666
Desvio Padrão	1,8163	1,7243	1,1527	1,1662	1,7026	1,6361	1,2405	1,1926	0,5375

Tabela 11 – Execuções do kacem 15x10 - Parte 2

Problema	15x10				
Mutação	5	5	5	5	5
Elitismo	20	20	20	20	20
Indivíduos	100	500	1000	2000	2000
Nº Gerações	500	500	100	100	100
Nº de Execuções	30	30	30	30	30
Menor Valor	17	12	12	12	12
Maior Valor	21	14	14	13	12
Média	18,233	12,8	12,666	12,1	12
Desvio Padrão	1,1743	0,5416	0,5374	0,300	0

Em resumo, os resultados acima apresentados, obtidos através do algoritmo proposto neste trabalho, apresentaram os valores de *makespan* 11, 15, 11, 7 e 12 para as amostras de teste (instâncias de Kacem) 4x5, 8x8, 10x7, 10x10 e 15x10, respectivamente.

Para as 3 primeiras instâncias totais, 4x5, 10x7 e 10x10, foram obtidas soluções ótimas, comparáveis a diversos outros trabalhos como Nouri, Driss e Ghédira (2015), Wang et al. (2013), Caldeira e Gnanavelbabu (2019), e valores de benchmarks compilados em Dauzère-Perès et al. (2024). Embora o resultado para a instância 15x10 não tenha atingido a solução ótima, que seria o valor 11 representando o melhor valor alcançado na literatura, o algoritmo demonstrou, de maneira geral, desempenho satisfatório, chegando ao valor 12, isto é, a apenas uma unidade de tempo do valor presente na literatura. É importante observar que a análise se restringiu exclusivamente às instâncias de Kacem, em contraste com outros estudos que exploraram diversos benchmarks como em Dauzère-Perès et al. (2024).

Tabela 12 – Comparativo Kacem, Hammadi e Borne (2002) e Carvalho (2015)

Problema	**EDAh**	**PSO+TS**	**PSO+SA**	**TS+VNS**	**MOGA**	**SEA**	**DIPSO**
KC 4x5	11	11	-	11	11	11	11
KC 8x8	15	14	15	14	15	14	14
KC 10x7	11	-	-	11	-	11	11
KC 10x10	7	7	7	7	7	7	7
KC 15x10	12	11	12	11	11	11	12

Na tabela 12, a coluna EDAh apresenta os valores obtidos na execução do algoritmo proposto neste trabalho de TCC (EDA-Híbrido) em comparação com outros trabalhos citados por Carvalho (2015).

Os melhores resultados para as instâncias Kacem, encontrados na literatura, são os mesmos apresentados na coluna **TS+VNS** e **SEA** da Tabela 12.

4.3 Análise dos Resultados

A análise dos dados obtidos com a execução do algoritmo sobre as matrizes ETC propostas por Kacem, Hammadi e Borne (2002) consistiram em verificar quais as combinações de parâmetros de entrada resultaram em melhores valores de makespan, como eles afetaram nesse resultado e a partir de que momento eles passaram a não mais acrescentar vantagens na variação dos valores.

De modo geral, à medida que se aumentava o número de indivíduos, melhores resultados eram encontrados. Isso pode ser explicado pelo fato de uma população maior fornecer uma diversidade maior de soluções iniciais, e com isso aumenta a probabilidade de, dentre essas soluções, algum indivíduo com melhores características ser encontrado e favorecer as demais etapas de mutação e cruzamento. Mas, a partir de certo ponto, após os valores convergirem para um valor mínimo do makespan, de nada se adiantava aumentar o número de indivíduos, já que o resultado não era alterado. Além disso, um impacto negativo percebido, e já esperado, foi que com o aumento do número de indivíduos o custo computacional também aumentava.

Com relação ao número de gerações, o mesmo fato ocorrido com a variação do número de indivíduos se repetiu. À medida que o número de gerações aumentava, menor o makespan se tornava. E, para cada combinação dos argumentos de entrada, um novo ponto de convergência era observado, e, a partir desse ponto, nenhuma melhoria no resultado era obtido. Esse comportamento pode ser explicado pelo fato do algoritmos não ter tempo de alcançar bons resultados quando a quantidade de gerações é menor, convergindo para um valor não ótimo. Por outro lado, quando a quantidade de gerações era significativa, mais profundamente o espaço de buscas era explorado e por sua vez o algoritmo tinha

a oportunidade de alcançar resultados melhores. Também com a variação para mais na quantidade de gerações, ocasionava um custo computacional maior.

Diferentemente dos dois argumentos anteriores, com relação à Mutação foi necessário encontrar um ponto de equilíbrio no qual melhores resultados eram obtidos. Pois, quando o percentual de mutação ficava muito baixo, não eram obtidos resultados satisfatórios, e por outro lado, quando se aumentava muito o valor em comparação ao ponto de equilíbrio encontrado, resultados mais discrepantes eram observados.

Novamente algumas são as razões para esse comportamento, já que um porcentual muito baixo de mutação faz com que haja a necessidade de se explorar uma população maior com mais diversidade ou então mais gerações de indivíduos. Mesmo tendo a vantagem de permitir que característica vantajosas ou benéficas se mantenha na população, enquanto um valor mais alto para mutação causa uma diversificação maior, muitas vezes alterando trechos do indivíduo que propiciavam características benéficas.

Quanto ao Elitismo, mais uma vez o valor com os melhores resultados foi obtido empiricamente, não sendo percebido nenhum resultado vantajoso para valores acima ou abaixo do valor ao qual forneceu o melhor makespan.

4.4 Discussão

Em relação aos resultados, o algoritmo proposto neste trabalho obteve os valores de *makespan* 11, 15, 11, 7 e 12 para as amostras de teste (instâncias de Kacem) 4x5 (tabela 6), 8x8 (tabela 7), 10x7 (tabela 8), 10x10 (tabela 9) e 15x10 (tabela 11), respectivamente. Para as 3 primeiras instâncias foram obtidas soluções ótimas, comparáveis a diversos outros estudos Wang et al. (2013), Caldeira e Gnanavelbabu (2019). Embora o resultado para a instância 15x10 não tenha atingido a solução ótima, com 11 representando o melhor valor alcançado na literatura, o algoritmo demonstrou, de maneira geral, desempenho satisfatório. É importante observar que a análise se restringiu exclusivamente às instâncias de Kacem, em contraste com outros estudos que exploraram diversos benchmarks como em Dauzère-Perès et al. (2024).

Os resultados satisfatórios, chancelaram a viabilidade de adaptação do algoritmo híbrido aplicado ao HCSP nos problemas FJSP, levando em consideração suas particularidades. Foi evidenciada a amplitude e aplicabilidade das técnicas de meta-heurísticas em contextos de escalonamento de elevada complexidade, nos quais métodos convencionais não conseguem produzir resultados satisfatórios Xie Liang Gao (2019). A utilização da estimativa de distribuição de probabilidades realizada em conjunto com os operadores genéticos foi um fator de importância para o sucesso dos resultados. O estudo de Cao et al. (2022) também demonstrou a eficácia e praticidade dessa abordagem na resolução de FJSP multiobjetivo.

Os bons resultados obtidos na execução do algoritmo híbrido, neste trabalho, corroboram com a ideia que, devido à complexidade intrínseca aos problemas de escalonamento e à ausência de algoritmos exatos que garantam soluções ótimas, a resolução eficaz desses desafios requer a implementação de técnicas não convencionais, especialmente quando há extensos volumes de tarefas a serem distribuídas Xie Liang Gao (2019). E, nesse cenário complexo específico, utilizou-se a meta-heurística EDA com operadores genéticos, destacando-se três operadores específicos - mutação, cruzamento e elitismo - como elementos fundamentais.

Considerando as atuais tendências de desenvolvimento na indústria de transformação e as aplicações FJSP do mundo real, este trabalho não apenas visou abordar desafios imediatos, mas também contribuir para as pesquisas relacionadas ao FJSP.

5 Conclusão

Os estudos sobre *Estimation Distribution Algorithm* (EDA) e *Heterogeneous Computing Scheduling Problem* (HCSP) foram essenciais para compreensão do problema que o algoritmo inicial fornecido propunha fazer, enquanto o estudo de *Flexible Job Shop Scheduling Problem* (FJSP) deu o norte a ser seguido na execução das alterações do algoritmo e na elaboração do novo código que buscava atender os objetivos propostos nesse trabalho.

A compreensão do algoritmo híbrido baseado em EDA aplicado ao HCSP, proporcionou uma visão mais esclarecedora sobre a implementação dessas meta-heurísticas em contextos de otimização.

Após a realização das alterações no algoritmo e elaboração de um novo código destinado aos problemas FJSP, pôde-se passar para a etapa de testes e coleta de resultados, que aqui, especificamente relacionados aos problemas de escalonamento *Flexible Job Shop Scheduling Problem* (FJSP), se mostraram adequados já que retornaram resultados satisfatórios, isto é, dentro do que é encontrado na literatura relacionada ao FJSP.

Outro ponto relevante foi demonstrar a possibilidade de adaptação do algoritmo híbrido aplicado ao HCSP nos problemas FJSP. Bastando levar em consideração suas peculiaridades. Assim, perceber a abrangência e aplicabilidade das técnicas de meta-heurísticas em problemas de escalonamento de difícil solução, em que métodos tradicionais não são capazes de fornecer resultados satisfatórios.

Uma das limitações que impediram uma aplicação mais abrangente, em especial a diferentes *benchmarks*, foi o fato de não ser possível trabalhar com problemas que possuam Matrizes de Tempo (MP) incompletas, isto é, que não possuam tempos de todos os *jobs* em todas as máquinas disponíveis. Além disso, a falta de tempo para explorar outros problemas foi significativa, uma vez que os ajustes e adaptações do código consumiram grande parte do prazo disponível.

Pela relevância e aplicabilidade do tema, sugere-se, para futuros trabalhos e investigações, estender a utilização do algoritmo híbrido baseado em EDA, desenvolvido neste trabalho, a diferentes benchmarks, em especial aqueles que possuam Matrizes de Tempos (MP) incompletas.

O algoritmo, utilizado nesse TCC e implementado em Python, está disponível para consulta e utilização pública no repositório do GitHub Borges (2024). O código-fonte é fornecido com os exemplos e testes utilizados, permitindo que pesquisadores e profissionais possam aplicar o algoritmo em seus próprios contextos e o modifiquem conforme necessário.

Referências

BORGES, C. **EDA híbrido para aplicação em FJSP**. 2024. Disponível em: <(https://github.com/celsoemiliano/EDAh>. Citado na página 44.

CALDEIRA, R. H.; GNANAVELBABU, A. Solving the flexible job shop scheduling problem using an improved jaya algorithm. **Computers Industrial Engineering**, v. 137, p. 106064, 2019. ISSN 0360-8352. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360835219305236>. Citado 2 vezes nas páginas 40 e 42.

CAO, L.; JIANG, M.; FENG, L.; LIN, Q.; PAN, R.; TAN, K. C. Hybrid estimation of distribution based on knowledge transfer for flexible job-shop scheduling problem. p. 1–6, 2022. Citado na página 42.

CARVALHO, L. C. F. **Algoritmo Híbido Multiobjetivo para o problema Flexible Job Shop**. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, Brasil, 2015. Disponível em: <https://repositorio.ufu.br/handle/123456789/12583>. Acesso em: 23 dez. 2022. Citado 4 vezes nas páginas 5, 18, 24 e 41.

CARVALHO, L. C. F. **Indicadores de Convergência e Diversidade em Algoritmos Evolutivos para Otimização Multiobjetivo**. Tese (Doutorado) — Universidade Federal de Uberlândia, Uberlândia, Brasil, 2021. Disponível em: <http://doi.org/10.14393/ufu.te.2021.511>. Acesso em: 23 dez. 2022. Citado 3 vezes nas páginas 10, 13 e 19.

DAUZÈRE-PERÈS, S.; DING, J.; SHEN, L.; TAMSSAOUET, K. The flexible job shop scheduling problem: A review. **European Journal of Operational Research**, v. 314, n. 2, p. 409–432, 2024. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S037722172300382X>. Citado 2 vezes nas páginas 40 e 42.

DRISS, K. N. M. I.; LAGGOUN, A. A new genetic algorithm for flexible job-shop scheduling problems. **Journal of Mechanical Science and Technology**, v. 29, n. 3, p. 1273–1281, 2015. Citado na página 17.

GAO, K.; CAO, Z.; ZHANG, L.; CHEN, Z.; HAN, Y.; PAN, Q. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. **IEEE/CAA JOURNAL OF AUTOMATICA SINICA**, v. 6, n. 1, p. 904–916, 2019. Citado 4 vezes nas páginas 9, 15, 16 e 17.

HUANG, X. B.; YANG, L. X. A hybrid genetic algorithm for multiobjective flexible job shop scheduling problem considering transportation time. **Int. J Intell. Comput. Cyber**, v. 12, n. 2, p. 154–174, 2019. Citado na página 17.

KACEM, I.; HAMMADI, S.; BORNE, P. Pareto-optimality approach for exible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. **Mathematics and Computers in Simulation**, v. 60, n. 3, p. 245–276, 2002. Citado 9 vezes nas páginas 3, 5, 23, 33, 34, 35, 36, 37 e 41.

LARRAÑAGA, J. A. L. P. **An Introduction to Probabilistic Graphical Models**. Boston, MA: Springer US, 2002. 27–56 p. ISBN 978-1-4615-1539-5. Disponível em: <https://doi.org/10.1007/978-1-4615-1539-5_2>. Citado na página 24.

MAIER, H.; RAZAVI, S.; KAPELAN, Z.; MATOTT, L.; KASPRZYK, J.; TOLSON, B. Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. **Environmental Modelling Software**, v. 114, p. 195–213, 2019. ISSN 1364-8152. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1364815218305905>. Citado na página 9.

MASSOBRIO, R.; DORRONSORO, B.; NESMACHNOW, S. Virtual savant for the heterogeneous computing scheduling problem. In: **2018 International Conference on High Performance Computing Simulation (HPCS)**. [S.l.: s.n.], 2018. p. 821–827. Citado na página 15.

NOURI, H. E.; DRISS, O. B.; GHéDIRA, K. Hybrid metaheuristics within a holonic multiagent model for the flexible job shop problem. **Procedia Computer Science**, v. 60, p. 83–92, 2015. ISSN 1877-0509. Knowledge-Based and Intelligent Information Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050915022346>. Citado na página 40.

PEREZ-RODRIGUEZ, A. H.-A. R. A hybrid estimation of distribution algorithm for flexible job-shop scheduling problems with process plan flexibility. **Applied Intelligence**, v. 48, n. 10, p. 3707–3734, 2018. Citado na página 18.

ROCHA, M. de A. **Análise de um Algoritmo de Estimação de Distribuição Híbrido no Problema de Escalonamento de Tarefas Independentes**. Tese (Doutorado) — Universidade Federal de Uberlândia, Uberlândia, Brasil, 2023. Disponível em: <https://repositorio.ufu.br/handle>. Acesso em: 23 dez. 2022. Citado 3 vezes nas páginas 20, 24 e 25.

TEEKENG ARIT THAMMANO, P. U. J. K. W. A new algorithm for flexible job-shop scheduling problem based on particle swarm optimization. **Artif Life Robotics**, v. 21, n. 1, p. 18–23, 2016. Citado na página 18.

VARELA, L. **Uma contribuição para o escalonamento da produção baseado em métodos globalmente distribuídos**. Tese (Doutorado) — Universidade do Minho, 2007. Disponível em: <https://repositorium.sdum.uminho.pt/handle/1822/7234>. Acesso em: 23 dez. 2023. Citado na página 9.

VIKHAR, P. A. Evolutionary algorithms: A critical review and its future prospects. p. 261–265, 2016. Citado 2 vezes nas páginas 9 e 12.

WANG, S.; WANG, L.; LIU, M.; XU, Y. An estimation of distribution algorithm for the multi-objective flexible job-shop scheduling problem. In: **2013 IEEE Symposium on Computational Intelligence in Scheduling (CISched)**. [S.l.: s.n.], 2013. p. 1–8. Citado 2 vezes nas páginas 40 e 42.

WANG, Y.; XIAO, N.; YIN, H.; HU, E.; ZHAO, C.; JIANG, Y. A two-stage genetic algorithm for large size job shop scheduling problems. **The International Journal of Advanced Manufacturing Technology**, v. 39, p. 813–820, 11 2008. Citado na página 15.

XIE LIANG GAO, K. P. X. L. H. L. J. Review on flexible job shop scheduling. **IET Collaborative Intelligent Manufacturing**, v. 1, n. 3, p. 67–77, 2019. Citado 3 vezes nas páginas 17, 42 e 43.