UNIVERSIDADE NOVE DE JULHO - UNINOVE
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA E GESTÃO DO CONHECIMENTO - PPGI

DAVID RIOS BETONI

ABORDAGEM DE BUSCA LOCAL ATIVA EM SOLUÇÕES NÃO ATRASADAS PARA O PROBLEMA DE JOB SHOP SCHEDULING UTILIZANDO ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS

São Paulo
2026

DAVID RIOS BETONI

ABORDAGEM DE BUSCA LOCAL ATIVA EM SOLUÇÕES NÃO ATRASADAS PARA O PROBLEMA DE JOB SHOP SCHEDULING UTILIZANDO ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS

Dissertação de mestrado apresentada ao programa de Pós-graduação em Informática e Gestão do Conhecimento da Universidade Nove de Julho - UNINOVE, como parte dos requisitos para a obtenção do título de Mestre em Informática e Gestão do Conhecimento.

Prof. Orientador: Dr. Fabio Henrique Pereira

São Paulo
2026

Betoni, David Rios.
Abordagem de busca local ativa em soluções não atrasadas para o problema de job shop scheduling utilizando algoritmo genético de chaves aleatórias. / David Rios Betoni. 2026.
79 f.
Dissertação (Mestrado)- Universidade Nove de Julho - UNINOVE, São Paulo, 2026.
Orientador (a): Prof. Dr. Fabio Henrique Pereira.
1. Algoritmo genético de chaves aleatórias. 2. Job shop scheduling. 3. Otimização combinatória. 4. Refinamento de soluções. 5. Soluções não atrasadas.
I. Pereira, Fabio Henrique. II. Título.

CDU 004

UNINOVE
Universidade Nove de Julho

PARECER – EXAME DE DEFESA

Parecer da Comissão Examinadora designada para o exame de Defesa do Programa de Pós-Graduação em Informática e Gestão do Conhecimento a qual se submeteu o aluno David Rios Betoni.

Tendo examinado o trabalho apresentado para obtenção do título de “Mestre em Informática e Gestão do Conhecimento”, com Dissertação intitulada ABORDAGEM DE BUSCA LOCAL ATIVA EM SOLUÇÕES NÃO ATRASADAS PARA O PROBLEMA DE JOB SHOP SCHEDULING UTILIZANDO ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS, a Comissão Examinadora considerou o trabalho:

(X) Aprovado

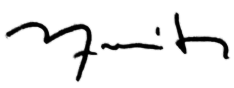
( ) Aprovado condicionalmente

( ) Reprovado com direito a novo exame

( ) Reprovado

Examinadores

Prof. Dr. Fabio Henrique Pereira



Prof. Dr. Marcone Jamilson Freitas Souza

Prof. Dr. Peterson Adriano Belan

segunda-feira, 2 de março de 2026

Dedico este trabalho principalmente à minha esposa Vanessa, pelo amor incondicional durante todo o tempo em que estamos juntos. À minha filha Sophia, para que você sempre acredite que o conhecimento é a chave que abre todas as portas. Que este trabalho seja um exemplo de dedicação para a sua própria jornada.

Agradecimentos

A Deus, por ser meu guia constante. Em muitos momentos, minha única prece era que, se fosse da Sua vontade, Ele me permitisse força e sabedoria para chegar ao final desta jornada. Hoje, ao concluir este trabalho, reconheço que Sua presença foi o alicerce que me manteve firme.

À minha família, minha esposa Vanessa e minha filha Sophia, que sempre estiveram ao meu lado, apoiando-me com amor, paciência e compreensão em cada etapa desta jornada acadêmica. O apoio de vocês foi a força motriz por trás de todas as minhas conquistas. Cada sacrifício feito não passou despercebido; sou eternamente grato pelo amor incondicional.

Aos meus pais, José Roberto e Geralda, e ao meu irmão Anderson, por terem me permitido crescer em um lar saudável e seguro.

Ao meu professor e orientador, Dr. Fábio Henrique Pereira, pela amizade, orientação, dedicação e, principalmente, pelo incentivo ao desenvolvimento deste trabalho. Serei eternamente grato por ter acreditado em mim.

Ao professor Dr. Peterson Belan, pela amizade, por ter me introduzido ao campo dos Algoritmos Genéticos e pelas conversas acolhedoras que tivemos nos corredores da universidade.

Aos demais professores do PPGI, que contribuíram para minha formação e para o desenvolvimento deste trabalho.

À Universidade Nove de Julho (UNINOVE), pela concessão da bolsa de estudos.

Aos amigos e colegas do meu círculo de convívio acadêmico, que me ajudaram no esclarecimento de dúvidas e pelo excelente trabalho em grupo.

Ao meu gerente na Genesys, Eduardo Marques, por permitir a flexibilização dos horários no trabalho, o que contribuiu imensamente para a concretização deste sonho. Jamais esquecerei esse apoio.

À CAPES, não apenas pelo apoio financeiro, mas também pela disponibilização do Portal de Periódicos, ferramenta indispensável para a consulta de referências internacionais e atualização do estado da arte sobre o Problema de Job Shop Scheduling.

Enfim, a todas as pessoas que incentivam os estudos e a busca pelo conhecimento.

"A simplicidade é o último grau de sofisticação."
Leonardo da Vinci

Resumo

Neste trabalho, apresenta-se uma abordagem computacional para a solução de problemas de sequenciamento de tarefas de produção, conhecidos como *Job Shop Scheduling*. Trata-se de um problema de otimização que busca definir a sequência ideal de operações para diferentes tarefas *(jobs)*, utilizando uma variedade de máquinas, com o objetivo de minimizar o tempo total de produção *(makespan)*. Na literatura atual, o uso de metaheurísticas tem sido amplamente difundido para resolver esse tipo de problema, em especial os Algoritmos Genéticos, por possibilitarem a obtenção de soluções de alta qualidade. Em geral, contudo, as metaheurísticas demandam o uso conjunto de técnicas de refinamento das soluções. Entre essas técnicas, destacam-se os métodos de Busca Local, que visam aprimorar os resultados obtidos pelas metaheurísticas. Este trabalho insere-se nesse contexto ao propor uma abordagem híbrida em duas etapas. Inicialmente, utiliza-se uma variação de Algoritmo Genético, denominada Algoritmo Genético de Chaves Aleatórias (RKGA), para gerar uma solução inicial de boa qualidade. Em seguida, após a conclusão do processo evolutivo, ou seja, sem incorporar mecanismos de busca local durante a execução do algoritmo genético, aplica-se uma etapa independente de refinamento baseada em Busca Local. Essa segunda etapa atua sobre a solução gerada pelo RKGA, explorando o espaço reduzido de soluções não atrasadas, com o objetivo de melhorar o tempo total de produção. Os resultados, obtidos a partir de conjuntos de problemas-teste padronizados e amplamente utilizados na literatura, mostram que soluções não atrasadas subótimas são frequentemente aprimoradas pelo método de refinamento proposto, alcançando resultados superiores aos da implementação convencional do algoritmo genético. Conclui-se, portanto, que os resultados obtidos validam a abordagem proposta, demonstrando sua eficácia na obtenção de soluções mais eficientes.

Palavras-chave: Algoritmo Genético de Chaves Aleatórias; Job Shop Scheduling; Otimização Combinatória; Refinamento de Soluções; Soluções Não Atrasadas

Abstract

In this work, a computational approach is presented for solving production task sequencing problems, known as *Job Shop Scheduling*. It is an optimization problem that seeks to define the ideal sequence of operations for different tasks *(jobs)* across a variety of machines, with the objective of minimizing the total production time *(makespan)*. In current literature, the use of metaheuristics has been widely adopted to solve this type of problem, especially Genetic Algorithms, as they enable the achievement of high-quality solutions. In general, however, metaheuristics require the joint use of solution refinement techniques. Among these techniques, Local Search methods stand out, aiming to improve the results obtained by the metaheuristics. This work fits into this context by proposing a two-stage hybrid approach. Initially, a variation of the Genetic Algorithm, known as the Random-Key Genetic Algorithm (RKGA), is used to generate a high-quality initial solution. Then, after the conclusion of the evolutionary process—that is, without incorporating local search mechanisms during the execution of the genetic algorithm—an independent refinement stage based on Local Search is applied. This second stage acts on the solution generated by the RKGA, exploring the reduced space of non-delay solutions, with the objective of improving the total production time. The results, obtained from standardized benchmark problem sets widely used in the literature, show that suboptimal non-delay solutions are frequently improved by the proposed refinement method, achieving results superior to those of the conventional genetic algorithm implementation. It is concluded, therefore, that the obtained results validate the proposed approach, demonstrating its effectiveness in achieving more efficient solutions.

Keywords:

Combinatorial Optimization; Job Shop Scheduling; Non-delay Schedules; Random-Key Genetic Algorithm; Solution Refinement

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AG	Algoritmo Genético
CPU	Central Processing Unit (Unidade Central de Processamento)
FT06	Instância *Benchmark* de Fisher e Thompson (1963)
GA	Genetic Algorithm (Algoritmo Genético)
Gantt	Diagrama de Gantt
ILS	Iterated Local Search (Busca Local Iterada)
JSSP	Job Shop Scheduling Problem (Problema de Sequenciamento Job Shop)
LA01/LA10	Benchmarks de Lawrence (1984)
NP-Hard	Problema de classe NP-difícil
PSO	Particle Swarm Optimization (Otimização por Enxame de Partículas)
RK	Random Key (Chave Aleatória)
RKGA	Random-Key Genetic Algorithm (Algoritmo Genético com Chaves Aleatórias)
SA	Simulated Annealing (Recozimento Simulado)
TS	Tabu Search (Busca Tabu)

Lista de Símbolos

$C_{\max}$	Makespan — tempo total necessário para concluir todas as *jobs*.
j	Índice representativo de uma *job*.
i	Índice representativo de uma operação ou máquina.
k	Índice auxiliar para operações ou iterações.
S_{ij}	Instante de início da operação i da *job* j.
C_{ij}	Instante de conclusão da operação i da *job* j.
p_{ij}	Tempo de processamento determinístico da operação i da *job* j.
O_{ij}	i-ésima operação da *job* j.
J	Conjunto de todas as *jobs* $J = \{J_1, J_2, \ldots, J_n\}$.
M	Conjunto de todas as máquinas $M = \{M_1, M_2, \ldots, M_m\}$.
n	Número total de *jobs*.
m	Número total de máquinas.
T_{jk}	Elemento da matriz de sequência tecnológica para a *job* j.
P_{jk}	Elemento da matriz de tempos de processamento para a *job* j.
P_t	Sequenciamento parcial contendo operações agendadas até a iteração t.
S_t	Conjunto de operações tecnologicamente disponíveis (candidatas) na iteração t.
ρ_k	Instante mais cedo possível para o início da operação $o_k \in S_t$.
b_k	Instante mais cedo de término da operação o_k.
b^*	Tempo mínimo de conclusão global entre as operações candidatas.
M^*	Máquina crítica onde ocorre o tempo de conclusão b^*.
Σ	Espaço de busca (conjunto de soluções factíveis).
$N(S)$	Vizinhança de uma solução S.
s	Vetor de atrasos iniciais por *job*, $s = [s_1, s_2, \ldots, s_n]$.
s_j	Tempo de atraso imposto ao início da *job* j.
s'	Vetor temporário de atrasos gerado durante a busca local.
d	Valor discreto de atraso testado.
slack	Folga entre operações consecutivas de uma *job*.
delay	Atraso aplicado para refinamento da solução.
fitness(s)	Função de aptidão, definida como $1/C_{\max}(s)$.
p_e	Fração de indivíduos de elite (5%).
p_m	Probabilidade de mutação (2%).
p_c	Taxa de cruzamento (*crossover*) (80%).
N_{pop}	Tamanho da população.
G_{max}	Número máximo de gerações.
$[0, 1)$	Intervalo contínuo para geração de chaves aleatórias.

Capítulo 1

Introdução

1.1 Contextualização do Problema

O *Job Shop Scheduling Problem (JSSP)* é um problema que consiste em organizar a sequência de operações de diferentes trabalhos *(jobs)* em um conjunto limitado de máquinas, buscando alcançar objetivos específicos, como a minimização do tempo total de produção, chamado de *makespan* (PINEDO, 2016). Este é um problema de otimização combinatória pertencente à classe de problemas NP-Difícil. Segundo (CORMEN et al., 2022), um problema NP-difícil significa que não se conhece um algoritmo que o resolva em tempo polinomial para todos os casos.

A dificuldade em encontrar soluções ótimas para estes tipos de problemas de grande porte, inerente à sua classificação NP-Difícil, reforça a necessidade de abordagens eficientes. Neste contexto, as metaheurísticas se destacam por guiar heurísticas na construção de algoritmos capazes de encontrar soluções de boa qualidade para problemas complexos, mesmo sem garantir a otimalidade (SöRENSEN; GLOVER, 2013).

Os Algoritmos Genéticos (AGs) têm se destacado por oferecer um bom desempenho na solução para este tipo de problema, devido à sua capacidade de pesquisa global (ROSA, 2019). Contudo, para melhorar os resultados é comum combinar a busca global com estratégias de Busca Local (HOLLAND, 1992). Essa integração, frequentemente caracterizada como abordagem híbrida ou algoritmo memético (BLICKLE; THIELE, 1996; EIBEN; SMITH, 2015), permite contornar limitações típicas dos AGs, como a convergência prematura (KATOCH; CHAUHAN; KUMAR, 2020; EIBEN; SMITH, 2015). A convergência prematura ocorre quando a população perde sua diversidade muito rapidamente. Nessa situação, o algoritmo passa a explorar apenas uma pequena região do espaço de soluções e acaba preso em uma solução que parece boa, mas que não é a melhor possível (o chamado ótimo local). Para evitar esse tipo de situação, é útil combinar o algoritmo genético com uma técnica de Busca Local, que atua como um mecanismo de refinamento. Enquanto o AG explora várias possibilidades de forma ampla, a Busca Local ajuda a melhorar soluções já promissoras, aumentando as chances de alcançar resultados mais próximos do ideal.

No contexto do JSSP, o tempo de espera é a diferença temporal entre o tempo de término de uma operação e o tempo de início da operação subsequente da mesma *job* em outra máquina é um indicador crítico da eficiência do sequenciamento. Este período é formalmente designado como tempo de espera da *job* (ou folga), e representa o intervalo em que a *job*, tendo concluído sua operação predecessora, precisa aguardar a liberação da máquina alocada para sua próxima operação. A ocorrência desse tempo de espera é reflexo direto dos conflitos de recursos entre *jobs* concorrentes e das restrições de prece-

dência tecnológica. Uma solução ótima minimiza tais tempos de espera, não os elimina necessariamente, pois são intrínsecos ao balanceamento entre a utilização das máquinas e o fluxo de trabalho de cada *job*. A proposta central deste trabalho reside na exploração estratégica dessa folga quando não se atinge o melhor conhecido. A ideia é reavaliar a solução obtida pelo Algoritmo Genético de Chaves Aleatórias *(Random-key Genetic Algorithm (RKGA))*, introduzindo tempos incrementados nas *jobs* que contêm as maiores folgas entre *jobs*. Tal movimentação controlada das *jobs* visa desalocar gargalos ou liberar recursos críticos, permitindo que outras *jobs* se movimentem em busca de um sequenciamento superior.

1.1.1 Justificativa e Contribuição do Estudo

Apesar dos bons resultados alcançados pelos Algoritmos Genéticos (AGs) na solução do *Job Shop Scheduling Problem (JSSP)*, ainda existe a necessidade de encontrar soluções melhores e de acelerar o processo de busca por essas soluções. Diante dessa importância, este estudo propõe uma adaptação do Algoritmo Genético de Chaves Aleatórias (RKGA) para reduzir o makespan. A principal ideia da proposta é acrescentar ao RKGA um método de Busca Local, que ajuda a melhorar as soluções já encontradas pelo algoritmo, tornando a busca mais eficiente. Com essa combinação, espera-se obter resultados de melhor qualidade do que aqueles gerados pelo RKGA tradicional, contribuindo para resolver o JSSP de forma mais eficaz.

1.2 Objetivo

Este estudo tem como objetivo principal o desenvolvimento e a avaliação de uma parametização do Algoritmo Genético de Chaves Aleatórias (RKGA), complementada por uma estratégia de Busca Local ativa, para a otimização do *makespan* no *Job Shop Scheduling Problem (JSSP)*.

1.2.1 Objetivos Específicos

1. Avaliar o impacto da estratégia de Busca Local ativa na redução do makespan em *benchmarks* clássicos do JSSP;

2. Comparar o desempenho da abordagem proposta com algoritmos genéticos sem o uso do refinamento proposto, em termos de qualidade da solução e tempo computacional para diferentes tamanhos de *benchmarks* do JSSP.

1.3 Organização da dissertação

Este trabalho está organizado em seis Capítulos, descritos a seguir:

- Capítulo 1: Introdução, contextualização, justificativa e objetivos do trabalho;
- Capítulo 2: Fundamentação teórica sobre otimização, o problema JSSP, Algoritmos Genéticos e Busca Local;
- Capítulo 3: Revisão da literatura sobre metaheurísticas aplicadas ao JSSP;
- Capítulo 4: Metodologia adotada;
- Capítulo 5: Apresentação e discussão dos resultados;
- Capítulo 6: Considerações finais e sugestões de trabalhos futuros.

Além dos Capítulos relacionados são apresentadas as referências bibliográficas.

Capítulo 2

Fundamentação teórica

Este capítulo tem como objetivo apresentar os fundamentos teóricos essenciais para a compreensão do presente trabalho. Serão abordados conceitos relacionados ao problema de sequenciamento de tarefas, o *JSSP*, Busca Local e classe de soluções. Além disso, será dedicada atenção especial às metaheurísticas, especificamente aos Algoritmos Genéticos, que compõem a base metodológica da abordagem proposta. A compreensão desses pilares teóricos é fundamental para a análise e validação dos resultados apresentados nas seções subsequentes.

2.1 Definição formal do JSSP

O *Job Shop Scheduling Problem (JSSP)* é um problema clássico da área de otimização combinatória. Ele envolve o sequenciamento de um conjunto de *(*Jobs) $J = \{J_1, J_2, \ldots, J_n\}$ sobre um conjunto M de máquinas, sendo que cada *(*Job) J_j consiste em uma sequência ordenada de operações $O_{j1}, O_{j2}, \ldots, O_{jm}$. Cada operação deve ser processada em uma máquina específica $M_k \in M$, por um tempo fixo p_{jk}.

O principal objetivo do problema é minimizar o tempo total necessário para completar todas as *(Jobs)*, conhecido como *makespan* ($C_{\max}$). Segundo a definição, a organização dessas tarefas obedece às seguintes restrições principais (PINEDO, 2016):

- Precedência Tecnológica: Cada *job* possui uma sequência linear e ordenada de operações (rota tecnológica) que deve ser respeitada, onde uma operação só pode iniciar após a conclusão da anterior na mesma *(*Job) .
- Capacidade das Máquinas: Cada máquina é um recurso limitado que pode processar apenas uma operação por vez, o que impede qualquer sobreposição temporal de tarefas distintas no mesmo equipamento.
- Exclusividade de Processamento da job: Uma *job* não pode ser processado em duas máquinas simultaneamente; ele deve ser finalizado em um recurso antes de ser transportado para o próximo no seu roteiro.
- Não-Preempção (Continuidade): Uma vez iniciado o processamento de uma operação, ele não pode ser interrompido, devendo prosseguir de forma contínua até o seu término total.
- Independência entre jobs: No modelo clássico, não existem restrições de precedência entre operações das *(*Jobs) diferentes, sendo a ordem de prioridade definida unicamente pelo algoritmo de sequenciamento.

- Disponibilidade Inicial: Assume-se que todas as *(*Jobs) e todas as máquinas estão disponíveis no instante zero ($t = 0$) para o início das atividades produtivas.
- Unicidade de Recurso e Roteiro: Cada operação deve ser realizada em uma única máquina específica previamente determinada, não existindo máquinas alternativas para a mesma tarefa no JSSP tradicional.

2.1.1 Representações do JSSP

Conforme proposto por (ROSA, 2019), utiliza-se o exemplo de (YAMADA, 2003) para ilustrar as distintas formas de representação do JSSP. O cenário, composto por três tarefas *(jobs)* e três máquinas, está detalhado na Tabela 2.1.

Tabela 2.1: *Exemplo de um problema JSSP com 3 jobs e 3 máquinas. Cada coluna representa a sequência tecnológica das máquinas para cada job com seu tempo de processamento entre parênteses.*

job	**Máquina (tempo de processamento)**		
J1	1(3)	2(3)	3(3)
J2	1(2)	3(3)	2(4)
J3	2(3)	1(2)	3(1)

Fonte: Traduzido de Yamada (2003).

Na Tabela 2.1 cada linha determina a sequência tecnológica de máquinas para a *job* correspondente e, o tempo de processamento da *job* naquela máquina é apresentado entre parênteses. A linha 1 corresponde as operações da *job* 1 que deverão ser realizadas na seguinte ordem: $O_{11} \longrightarrow O_{12} \longrightarrow O_{13}$, ou seja, a *job* 1 é processada primeiro na máquina 1 e seu tempo de processamento é igual a 3, depois na máquina 2 com tempo de processamento igual a 3 e por último, na máquina 3 com tempo de processamento igual a 3. Para os demais *jobs* apresentados no exemplo, segue-se a mesma lógica: *job* 2: $O_{21} \longrightarrow O_{23} \longrightarrow O_{22}$; e *job* 3: $O_{32} \longrightarrow O_{31} \longrightarrow O_{33}$.

2.1.2 Representação Matricial

A sequência tecnológica predefinida para cada *job* pode ser dada como uma matriz $\{T_{jk}\}$, na qual cada elemento $t_{jk} = i$ corresponde à k-ésima operação da *job* j na máquina i, denotada por O_{ji} (YAMADA, 2003).

À partir dos dados apresentados na Tabela 2.1, o problema pode ser representado pela matriz de sequência, ou matriz de operações dos *jobs* $\{T_{jk}\}$ e, pela matriz de tempos de processamento $\{P_{jk}\}$ conforme ilustra a Figura 2.1 (ROSA, 2019).

$$\{T_{jk}\} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix} \qquad \{P_{jk}\} = \begin{bmatrix} 3 & 3 & 3 \\ 2 & 3 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

Matriz de sequência T_{jk} das jobs, e matriz dos tempos de processamento P_{jk} para o problema 3×3 apresentado na Tabela 2.1. O elemento $t_{jk} = i$ representa que a k-ésima operação da job J_j deve ser processado na máquina M_i por p_{ji} unidades de tempo.

2.1.3 Representação por gráfico Gantt

O gráfico de Gantt representado na Figura 2.1 é uma ferramenta visual que ajuda a entender como as máquinas são usadas ao longo do tempo. Ele mostra quando cada operação acontece em cada máquina e permite ver o tempo total necessário para concluir todas as atividades. O eixo da abscissa representa a unidade de tempo e o eixo da ordenada representa as máquinas ou recursos. Cada retângulo no gráfico representa uma operação que é atribuída a uma máquina em um determinado instante de tempo. A largura do retângulo corresponde ao tempo de processamento da operação na máquina (GAO et al., 2015).

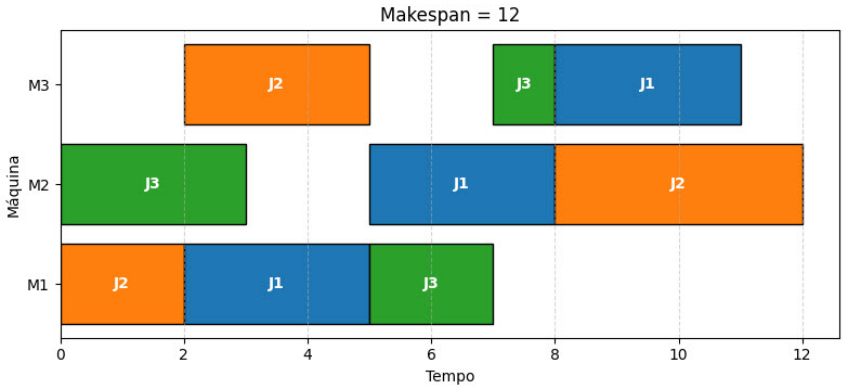


Figura 2.1: *Gráfico de Gantt para representação do JSSP*

2.1.4 Formulação Matemática do Problema de *Job Shop* Clássico

O problema do sequenciamento de trabalhos (*Job Shop Scheduling Problem* - JSSP) consiste em agendar um conjunto de n tarefas (*jobs*) em um conjunto de m máquinas (MORALES; RONCONI, 2016). Na versão clássica do problema, o sequenciamento deve respeitar estritamente a ordem de processamento de cada tarefa, garantir que uma máquina processe apenas uma tarefa por vez e considerar a não-preempção, ou seja, uma operação iniciada não pode ser interrompida (MORALES; RONCONI, 2016; BARAK; JAVANMARD; MOGHDANI, 2024).

Para encontrar a solução exata desse problema, a abordagem de Programação Linear Inteira Mista (PLIM) mais consolidada na literatura é o modelo disjuntivo de Manne (1960) (MORALES; RONCONI, 2016). Análises matemáticas demonstram que esta formulação apresenta um excelente desempenho computacional por exigir o menor número de variáveis binárias em comparação com outros modelos de designação da literatura (MORALES; RONCONI, 2016).

A formulação matemática clássica para a minimização do tempo máximo de conclusão (*makespan*) é descrita a seguir (MORALES; RONCONI, 2016; BARAK; JAVANMARD; MOGHDANI, 2024).

Conjuntos e Parâmetros

- $J = \{1, 2, \ldots, n\}$: Conjunto de tarefas (*jobs*) (MORALES; RONCONI, 2016).
- $M = \{1, 2, \ldots, m\}$: Conjunto de máquinas disponíveis (MORALES; RONCONI, 2016).
- p_{ij}: Tempo de processamento determinístico da tarefa $j \in J$ na máquina $i \in M$ (MORALES; RONCONI, 2016).
- σ_j: Ordem tecnológica de execução da tarefa j. A função $\sigma_j(k)$ retorna a k-ésima máquina exigida no roteiro da tarefa j (MORALES; RONCONI, 2016).
- P: Uma constante positiva suficientemente grande (conhecida como *Big-M*). Um limitante seguro adotado para P é o somatório de todos os tempos de processamento de todas as operações do problema (MORALES; RONCONI, 2016).

Variáveis de Decisão

- $C_{max} \geq 0$: Variável contínua que representa o *makespan*, isto é, o instante de conclusão da última tarefa processada no sistema (MORALES; RONCONI, 2016).
- $x_{ij} \geq 0$: Variável contínua que representa o tempo de início exato da tarefa j na máquina i (MORALES; RONCONI, 2016).
- $z_{ijk} \in \{0, 1\}$: Variável binária disjuntiva que assume o valor 1 se a tarefa j precede a tarefa k na máquina i, e o valor 0 caso contrário. É definida para todo par de tarefas onde $j < k$ (MORALES; RONCONI, 2016).

Modelo Matemático

O objetivo fundamental da formulação é minimizar o tempo de conclusão da última tarefa (MORALES; RONCONI, 2016; BARAK; JAVANMARD; MOGHDANI, 2024):

$$\text{Minimizar } C_{max} \tag{2.1}$$

Sujeito às seguintes restrições:

1. Restrições de Precedência (Roteamento): Garantem que a operação de uma tarefa em sua k-ésima máquina no roteiro só inicie após a conclusão da sua operação na máquina anterior $(k-1)$ (MORALES; RONCONI, 2016; BARAK; JAVANMARD; MOGHDANI, 2024).

$$x_{\sigma_j(k),j} \geq x_{\sigma_j(k-1),j} + p_{\sigma_j(k-1),j} \quad \forall j \in J, \forall k \in \{2, \ldots, m\} \tag{2.2}$$

2. Restrições Disjuntivas (Capacidade das Máquinas): Asseguram que uma máquina não processe duas tarefas (j e k) ao mesmo tempo. Através do uso da constante de penalidade P (*Big-M*) e da variável binária z_{ijk}, força-se a decisão de que ou a tarefa j precede a tarefa k, ou vice-versa (MORALES; RONCONI, 2016; BARAK; JAVANMARD; MOGHDANI, 2024).

$$x_{ik} \geq x_{ij} + p_{ij} - P(1 - z_{ijk}) \quad \forall j, k \in J \text{ com } j < k, \ \forall i \in M \tag{2.3}$$

$$x_{ij} \geq x_{ik} + p_{ik} - P \cdot z_{ijk} \quad \forall j, k \in J \text{ com } j < k, \ \forall i \in M \tag{2.4}$$

3. Restrição de Makespan: Obriga que a variável C_{max} seja maior ou igual ao instante de término do processamento da última tarefa. Para cada tarefa j, a última máquina de sua rota é representada por $\sigma_j(m)$ (MORALES; RONCONI, 2016).

$$C_{max} \geq x_{\sigma_j(m),j} + p_{\sigma_j(m),j} \quad \forall j \in J \tag{2.5}$$

4. Domínio das Variáveis: Define a não-negatividade para as variáveis contínuas e a natureza binária das variáveis disjuntivas (MORALES; RONCONI, 2016).

$$x_{ij} \geq 0 \quad \forall j \in J, \ \forall i \in M \tag{2.6}$$

$$C_{max} \geq 0 \tag{2.7}$$

$$z_{ijk} \in \{0, 1\} \quad \forall j, k \in J \text{ com } j < k, \ \forall i \in M \tag{2.8}$$

2.1.4.1 Função Objetivo

O objetivo do Problema de Job Shop Scheduling é minimizar o makespan (PINEDO, 2016; FISHER; THOMPSON, 1963), ou seja, o tempo total necessário para completar todas as *(jobs)*, sendo representado por:

$$\min C_{\max}. \tag{2.9}$$

2.2 Classes de Soluções no JSSP

O espaço de busca para no JSSP pode ser segmentado em três subconjuntos principais de sequenciamentos factíveis, organizados de forma hierárquica: soluções semi-ativas, ativas e não atrasadas (PINEDO, 2016; SILVA et al., 2012; JÚNIOR; ROSA; PEREIRA, 2019). A imagem 2.2 mostra a relação entre os diferentes tipos de sequenciamentos.

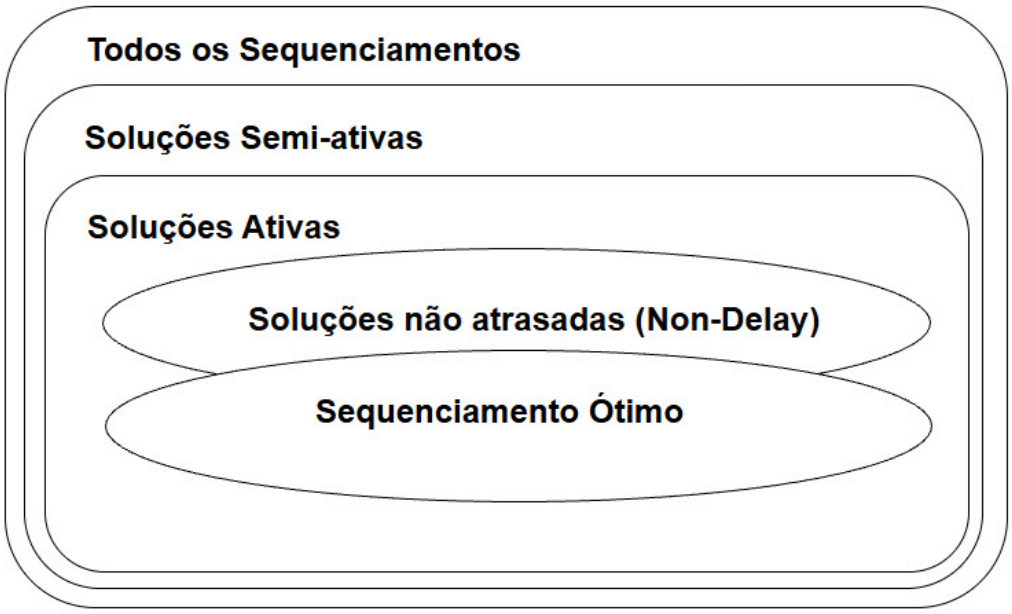


Figura 2.2: *Tipos de soluções*

Fonte: adaptado de (MOONEN; JANSSENS, 2007)

2.2.1 Soluções semi-ativas

Um sequenciamento é classificado como semi-ativo quando não apresenta tempos de espera desnecessários, o que significa que nenhuma operação pode ser iniciada mais cedo sem que a ordem de processamento nas máquinas seja alterada (PINEDO, 2016; GONÇALVES; MENDES; RESENDE, 2005). Nesta classe, as operações são agendadas o mais

cedo possível, respeitando apenas as restrições de precedência tecnológica (SILVA et al., 2012).

2.2.2 Soluções ativas

As soluções ativas formam uma subclasse das soluções semi-ativas e são definidas pela propriedade fundamental de que não é possível antecipar o término de qualquer operação sem que isso resulte no atraso de pelo menos uma outra tarefa ou na violação das restrições de precedência (PINEDO, 2016; JÚNIOR; ROSA; PEREIRA, 2019). É formalmente provado que, para funções objetivo regulares como a minimização do *makespan* ($C_{\max}$), o conjunto das soluções ativas sempre contém pelo menos uma solução ótima (BOUKEDROUN et al., 2023; PINEDO, 2016).

2.2.3 Soluções não atrasadas *(Non-Delay)* no JSSP

As soluções não atrasadas (*Non-Delay Schedules*) representam uma classe de sequenciamento no JSSP onde nenhuma máquina é mantida ociosa se houver *(*Job) disponível (BEAN, 1994; LONDE et al., 2025). Esta classe é uma subclasse das soluções ativas e constitui o menor espaço de busca viável, o que favorece a eficiência computacional de metaheurísticas (GONÇALVES; MENDES; RESENDE, 2005; JUNIOR, 2015).

Apesar de simplificar o processo de busca, restringir o algoritmo a soluções *Non-Delay* pode resultar na perda da solução ótima global, uma vez que a solução ótima nem sempre está contida neste subconjunto (LONDE et al., 2025; CONSTANTINO; SEGURA, 2021). Para contornar essa limitação, pesquisadores utilizam estratégias de hibridização com Busca Local e o monitoramento da diversidade populacional são essenciais para evitar ótimos estritamente locais e a convergência prematura (RAFSANJANI; RIYAHI, 2020; SUDHOLT, 2016; BLICKLE; THIELE, 1996).

2.2.4 Busca Local

A Busca Local (*Local Search* - LS) é uma estratégia heurística fundamental utilizada para resolver Problemas de Otimização Combinatória, especialmente aqueles que são NP-difíceis (LONDE et al., 2025; JúNIOR, 2021). Diferentemente de outros algoritmos que constroem uma solução do zero, a Busca Local é um processo iterativo que visa a exploração e refinamento do espaço de soluções de um problema (LONDE et al., 2025).

A Busca Local parte de uma solução inicial, que pode ser gerada de forma aleatória ou através de uma heurística construtiva (LONDE et al., 2025). A partir dessa solução, o algoritmo tenta encontrar soluções de melhor qualidade em sua vizinhança (*neighborhood*) (LONDE et al., 2025).

O nome "Busca Local" deriva do fato de que a exploração do espaço de soluções é limitada a um subconjunto específico de soluções vizinhas. Em um problema de otimização, dado o espaço de busca Σ (o conjunto de soluções factíveis), uma função de vizinhança N é um mapeamento $N : \Sigma \to 2^{\Sigma}$ que define, para cada solução $S \in \Sigma$, um subconjunto $N(S) \subset \Sigma$ chamado vizinhança (LONDE et al., 2025).

Cada solução $S' \in N(S)$ é um vizinho de S, obtido através de um movimento (*move*) (LONDE et al., 2025; EIBEN; SMITH, 2015), que corresponde a uma pequena alteração estrutural feita na solução corrente. A escolha da função de vizinhança é um fator crucial que afeta tanto a eficiência quanto a eficácia da Busca Local. Movimentos típicos incluem permutações (*swap*) ou inserções de elementos, como a troca de duas jobs em um sequenciamento (LONDE et al., 2025; JúNIOR, 2021).

O algoritmo de Busca Local fica inspecionando a vizinhança da solução atual e a substitui pelo melhor vizinho encontrado, em um processo que continua até que nenhum vizinho $S' \in N(S)$ seja encontrado com melhor valor para a função objetivo (LONDE et al., 2025).

Embora a Busca Local seja frequentemente rápida em encontrar boas soluções, ela apresenta a desvantagem de tender a se tornar "presa" em ótimos estritamente locais (LONDE et al., 2025). Caso o algoritmo aceite apenas soluções que melhoram a atual, ele não consegue escapar desses mínimos locais, o que pode resultar em uma solução significativamente inferior ao ótimo global (LONDE et al., 2025).

Para mitigar esse risco e promover a intensificação da busca, a Busca Local pode ser utilizada como um componente de refino em metaheurísticas mais complexas, como o RKGA. No contexto deste trabalho a Busca Local é utilizada como uma etapa complementar ao algoritmo genético: primeiro o RKGA encontra uma boa solução para o problema de sequenciamento e, em seguida, a Busca Local tenta melhorar essa solução ajustando pontualmente o sequenciamento através de folgas entre as *jobs*, explorando pequenas modificações que possam reduzir ainda mais o tempo total de conclusão (*makespan*), sem alterar toda a estrutura do método.

2.3 Definição de Algoritmo Genético (AG)

O Algoritmo Genético (AG) é um método de busca e otimização pertencente à classe dos algoritmos de computação evolucionária (EIBEN; SMITH, 2015). Inspirado no processo de evolução biológica natural (MITCHELL, 1996; KATOCH; CHAUHAN; KUMAR, 2020; HOLLAND, 1992), ele se baseia na Teoria Darwiniana da "sobrevivência do mais apto" (survival of the fittest) (MITCHELL, 1996; GOLDBERG, 1989).

Proposto por John H. Holland em 1975, cujo trabalho foi posteriormente expandido e destacado e popularizado por (GOLDBERG, 1989), o AG busca compreender o fenômeno da evolução e aplicar seus princípios de adaptação em sistemas computacionais

(KATOCH; CHAUHAN; KUMAR, 2020).

Os Algoritmos Genéticos (AG) operam sobre uma população de soluções potenciais, denominadas indivíduos ou cromossomos (GOLDBERG, 1989; EIBEN; SMITH, 2015). Essa população evolui ao longo de sucessivas gerações por meio de operadores inspirados na seleção natural, como seleção, cruzamento (*crossover*) e mutação (MITCHELL, 1996; EIBEN; SMITH, 2015; LONDE et al., 2025). Isso garante que os indivíduos mais aptos, tenham maior probabilidade de sobreviver e propagar suas características genéticas.

No contexto do *Job Shop Scheduling Problem* (JSSP), cada cromossomo representa uma codificação de um sequenciamento, frequentemente estruturado como uma sequência de tarefas a serem executadas em cada máquina. O processo evolutivo mantém diversas soluções simultâneas no espaço de busca, refinando-as iterativamente até a convergência para um sequenciamento de alta qualidade ou ótimo.

Estrutura e Funcionamento:

1. O AG é um algoritmo de busca baseado em população *(population based search algorithm)* (GOLDBERG, 1989), operando simultaneamente sobre um conjunto de possíveis soluções (a população de cromossomos) (SUDHOLT, 2016).

2. Uma solução potencial é tratada como um indivíduo (MITCHELL, 1996) ou cromossomo (cadeia de bits, números inteiros ou reais), que codifica uma solução candidata para o problema (um ponto no espaço de busca) (MITCHELL, 1996; EIBEN; SMITH, 2015).

3. O algoritmo funciona através de um processo iterativo, onde a cada geração, novos conjuntos de aproximações são criados (KATOCH; CHAUHAN; KUMAR, 2020).

4. A qualidade de um indivíduo é determinada pela função *fitness* (ou função de aptidão), que avalia a performance do cromossomo (MITCHELL, 1996; BEASLEY, 1993; EIBEN; SMITH, 2015).

5. Os mecanismos evolucionários que impulsionam a busca são os operadores genéticos: seleção, cruzamento (*crossover*) e mutação (MITCHELL, 1996; GOLDBERG, 1989; BäCK, 1996; EIBEN; SMITH, 2015). Estes operadores transformam a população, gerando diversidade e mantendo as características de adaptação adquiridas (SHYLO et al., 2020).

Devido à sua simplicidade, flexibilidade e robustez, o AG tem sido usado para resolver problemas nos quais outras técnicas de otimização enfrentam obstáculos (EIBEN; SMITH, 2015). Na Figura 2.3 está uma demonstração visual de como são as etapas do funcionamento de um AG.

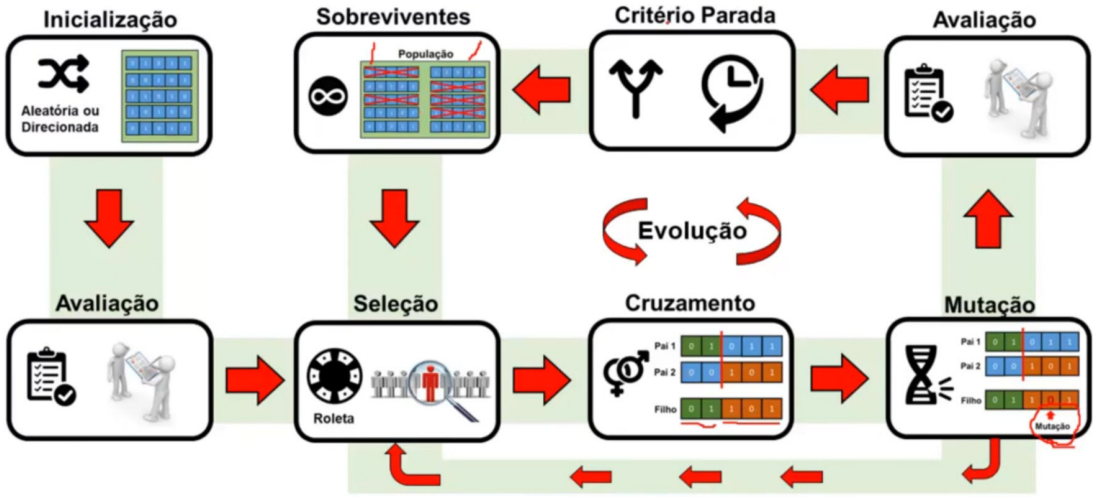


Figura 2.3: *Funcionamento do Algoritmo Genético*

Fonte: (PULIN, 2021)

2.4 Algoritmo de Giffler e Thompson

O algoritmo de Giffler e Thompson (GT), proposto originalmente em 1960, é um método construtivo fundamental na resolução do problema de *Job Shop Scheduling*. O seu principal objetivo é gerar sequenciamentos ativos, uma classe de soluções onde nenhuma operação pode ser antecipada na programação sem que isso resulte no atraso de outra operação ou na violação de restrições de precedência (PINEDO, 2016; MOONEN; JANSSENS, 2007).

A importância deste algoritmo reside no fato de que o espaço de busca das soluções ativas contém, garantidamente, a solução ótima para o problema de minimização do *makespan* ($C_{\max}$). O procedimento constrói o sequenciamento passo a passo, resolvendo conflitos entre operações que competem pelo mesmo recurso (máquina) simultaneamente (GONÇALVES; MENDES; RESENDE, 2005).

2.4.1 Funcionamento e Estrutura Lógica

O procedimento opera de forma iterativa. A cada passo, define-se um conjunto de operações disponíveis para serem agendadas (aquelas cujos predecessores tecnológicos já foram concluídos). A decisão crítica do algoritmo é determinar qual operação agendar a seguir para evitar tempos de inatividade desnecessários nas máquinas (MOONEN; JANSSENS, 2007).

A notação utilizada para descrever o algoritmo clássico, baseada em (MOONEN; JANSSENS, 2007), é definida a seguir:

- P_t: Sequenciamento parcial contendo as operações já agendadas até a iteração t.
- S_t: Conjunto de operações tecnologicamente disponíveis na iteração t (candidatas).

- ρ_k: Instante mais cedo possível para o início da operação $o_k \in S_t$, respeitando o término do seu antecessor e a disponibilidade da máquina.
- b_k: Instante mais cedo de término da operação o_k, dado por $b_k = \rho_k + p_k$, onde p_k é o tempo de processamento.

A execução do algoritmo segue cinco etapas principais que garantem a atividade do sequenciamento gerado, e que está ilustrado através do Figura do fluxograma 2.4 em seguida:

Passo 1: Inicialização: No instante $t = 1$, o sequenciamento parcial P_1 inicia-se vazio. O conjunto S_1 é preenchido com todas as operações iniciais de cada tarefa (aquelas sem predecessores).

Passo 2: Determinação da Máquina Crítica: Calculam-se os tempos de término mais cedo (b_k) para todas as operações em S_t. Identifica-se o tempo mínimo de conclusão global (b^*) e a máquina onde ele ocorre (M^*):

$$b^* = \min_{o_k \in S_t} \{b_k\} \tag{2.10}$$

A máquina M^* é denominada máquina crítica para a iteração atual.

Passo 3: Formação do Conjunto de Conflito: Para garantir a propriedade de sequenciamento ativo, não se escolhe necessariamente a operação com o tempo b^*. Em vez disso, cria-se um **Conjunto de Conflito** contendo todas as operações candidatas $o_j \in S_t$ que utilizam a máquina M^* e cujo início é possível antes do término crítico b^*. A condição é dada por:

$$\rho_j < b^* \tag{2.11}$$

Essa etapa assegura que a máquina M^* não fique ociosa desnecessariamente à espera da operação que define b^*, caso outra operação possa iniciar antes.

Passo 4: Seleção e Atualização: Uma operação o^* é selecionada de dentro do Conjunto de Conflito (utilizando uma regra de prioridade ou escolha aleatória).

- A operação o^* é movida de S_t para o sequenciamento fixo P_{t+1}.
- A máquina M^* tem sua disponibilidade atualizada.
- A sucessora imediata de o^* (se houver) é adicionada ao conjunto de disponíveis S_{t+1}.

Passo 5: Iteração: Incrementa-se $t \leftarrow t + 1$. Se o conjunto S_t não estiver vazio, retorna-se ao Passo 2. Caso contrário, o algoritmo termina com um sequenciamento completo e ativo.

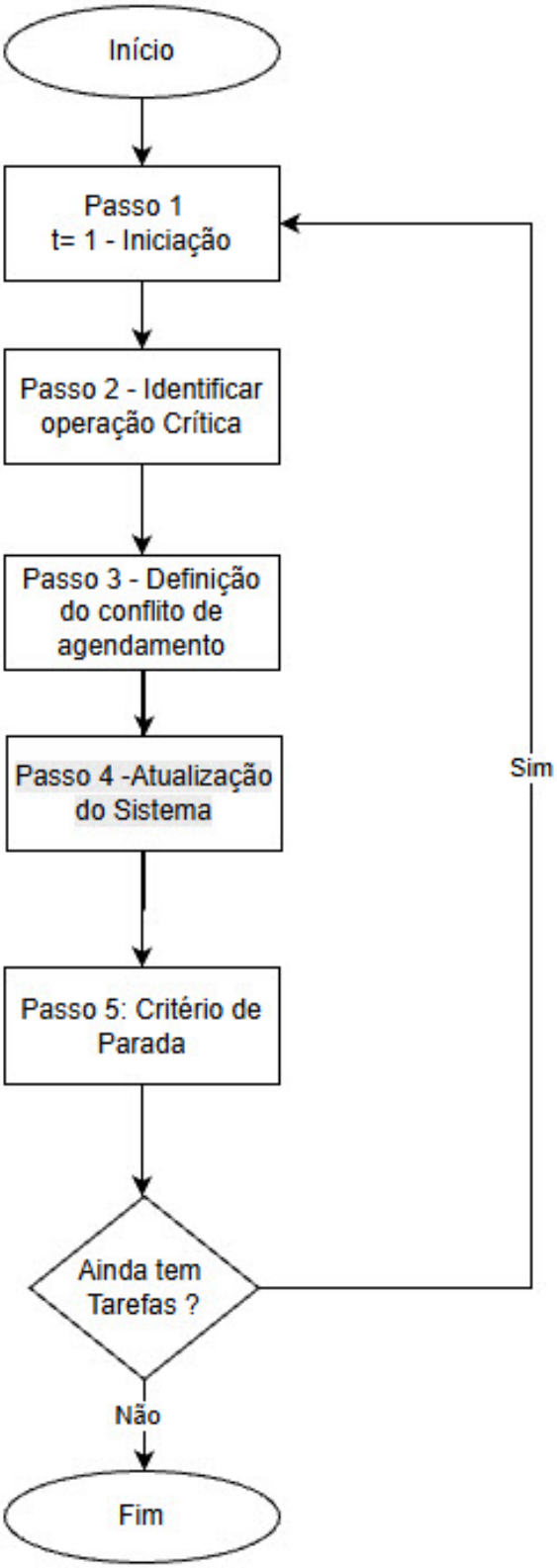


Figura 2.4: *Funcionamento da codificação Giffler–Thompson com suporte a atrasos estratégicos.*

Fonte: Adaptado de (MOONEN; JANSSENS, 2007).

Capítulo 3

Revisão da Literatura

Esta seção apresenta a revisão da literatura relacionada às diferentes abordagens propostas para a solução do JSSP. A análise deste problema é vasta e, dada a sua complexidade, diversos métodos e contribuições teóricas têm sido propostos por autores ao longo dos últimos anos. Com base em uma busca abrangente nas principais bases de conhecimento e publicações especializadas, este capítulo discute as contribuições mais recentes sobre o tema, com um foco particular em modelos que utilizam o Algoritmo Genético (AG) como base. O AG tem se destacado nesta área devido à sua robustez, capacidade intrínseca de lidar com problemas classificados como NP-difíceis, sua eficiência na exploração do espaço de soluções e sua adaptabilidade para trabalhar com múltiplos objetivos de otimização

3.1 Bases de Dados e Estratégia de Busca

A busca foi realizada nas seguintes bases de dados eletrônicas: Google Acadêmico, Portal de Periódicos Capes, SCIELO e SCOPUS. As palavras-chave utilizadas incluíram: " Job Shop Scheduling", "JSSP", " Job Shop Scheduling + local search", "genetic algorithm + random key", " Job Shop Scheduling + Busca Local + algoritmo genético" entre outras variações. Também foi realizada a pesquisa na Biblioteca Digital de Teses e Dissertações da Uninove, onde foi realizada a busca para o tema JSSP em estudos orientados na instituição. Artigos originais que estudaram estas abordagens de otimização de JSSP foram selecionados principalmente os trabalhos dos últimos dez anos. Estas buscas foram realizadas no decorrer do ano de 2025.

3.2 Critérios de Inclusão e Exclusão

Para este processo de seleção, a revisão foi realizada por um único revisor, o próprio autor da dissertação. Os critérios de inclusão e exclusão foram bem definidos. Para serem incluídos, os artigos deveriam focar no JSSP e utilizar algoritmos genéticos ou suas variações como técnica principal de resolução, ter sido publicados até 1º de julho de 2025. Foram analisados publicações no idioma Inglês e também no idioma Português do Brasil. Por outro lado, foram excluídos trabalhos não tinham um foco computacional, não utilizavam algoritmos genéticos, "ou que tratavam de outras variações.

3.3 Etapas de Seleção dos Estudos

O processo de seleção foi conduzido em quatro etapas: (1) identificação, (2) triagem por título e resumo, (3) leitura completa e (4) inclusão ou exclusão. O fluxograma do processo está representado na Figura 3.1.

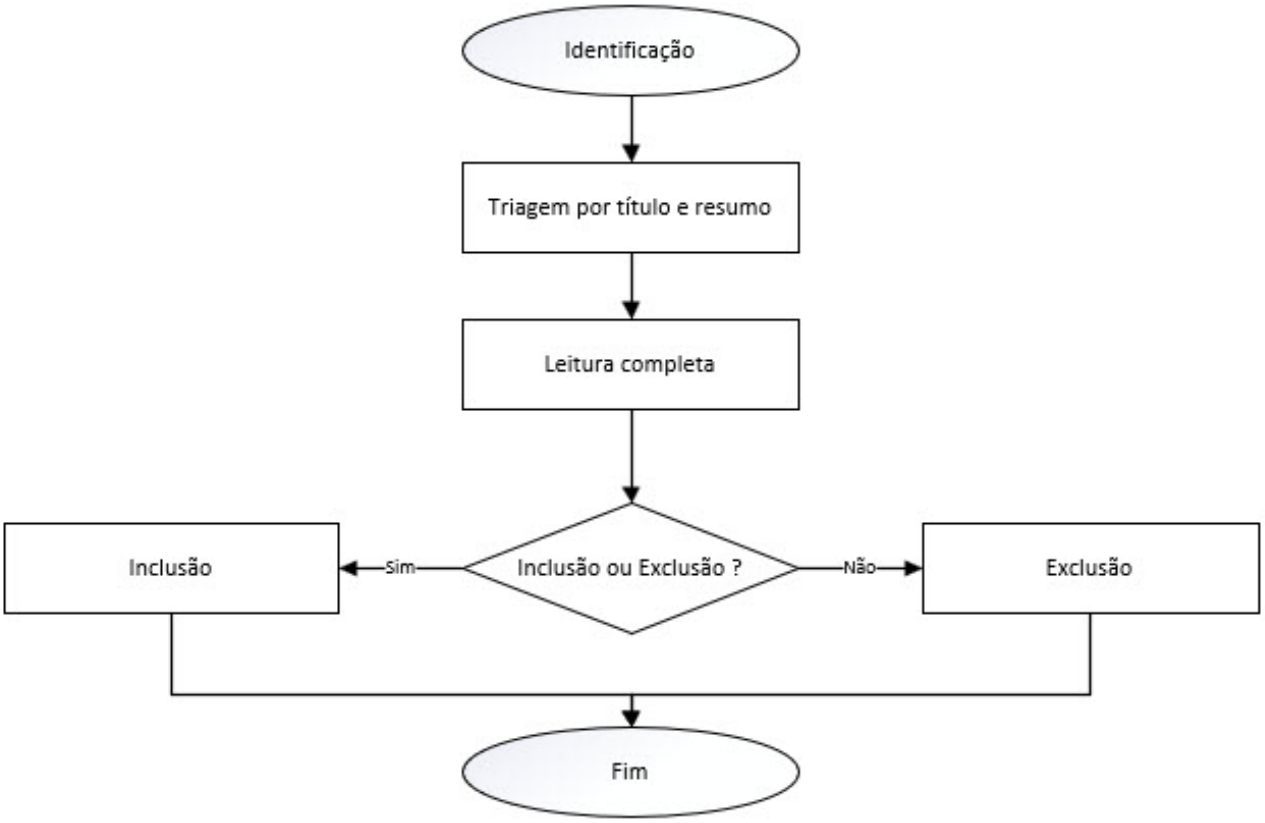


Figura 3.1: *Fluxograma do processo de seleção de estudos*

Fonte: Autor

3.4 Aplicação de técnicas metaheurísticas ao JSSP

Os métodos de otimização aplicados ao Problema de Job Shop Scheduling (JSSP) vêm sendo extensivamente estudados ao longo das últimas décadas, em especial as metaheurísticas baseadas em Algoritmos Genéticos e seus diversos mecanismos de refinamento. A literatura apresenta uma ampla gama de estratégias destinadas a melhorar o desempenho desses algoritmos, seja por meio da adaptação de operadores, da introdução de representações cromossômicas eficientes ou da combinação entre busca global e Busca Local.

As abordagens verificadas, foram baseadas em metaheurísticas e representam soluções de diferentes maneiras, desde listas de prioridades para todas as operações até representações binárias ou reais, baseadas em chaves aleatórias . Além disso, esses métodos utilizam valores fixos para o limite de tempo ocioso aceitável ou exigem um procedimento adicional para otimizar esse parâmetro para criar sequenciamentos ativos parametrizados. Os autores em (PALACIOS et al., 2014; PETROVIC et al., 2013), por exemplo, definiram um algoritmo genético no qual cada cromossomo tem o tamanho 2n, onde n é o número de operações do problema. A primeira metade do cromossomo representa valores de prioridade e a segunda metade define os tempos ociosos para cada operação. Apesar dos bons resultados obtidos por esses autores, essa abordagem aumenta o espaço de busca e, possivelmente, o tempo computacional. Muitas abordagens são baseadas em ideias semelhantes.

Um dos principais temas de pesquisa abordados é a hibridização de AGs com outras técnicas de Busca Local, como visto nos trabalhos de (GONÇALVES; MENDES; RESENDE, 2005) e (CONSTANTINO; SEGURA, 2021). Esses métodos combinam a capacidade de exploração global dos AGs com o refinamento local, aprimorando a qualidade da solução. (VIANA; JUNIOR; CONTRERAS, 2020) também abordam essa questão, focando em operadores de mutação e cruzamento aprimorados para evitar a convergência prematura.

Referente a análise da estrutura do espaço de busca para guiar o processo evolutivo, o trabalho de (ROSA, 2019) apresenta uma abordagem inovadora ao propor um Algoritmo Genético Híbrido (HGA) fundamentado na Análise de Componentes Principais (PCA) do *fitness landscape*. Diferente de abordagens puramente estáticas, a autora utiliza a PCA para monitorar a diversidade populacional de forma *online*, classificando os indivíduos por semelhança e calculando um índice individual de contribuição para a diversidade (ROSA, 2019). Esta estratégia permite identificar momentos de estagnação e substituir soluções redundantes por meio de uma etapa de intensificação que combina um Algoritmo Genético Binário Bidimensional (GAB) com o método *Path-relinking*, visando explorar a estrutura de "grande vale"(*big valley*) do espaço de busca para alcançar o *makespan* melhor conhecido (ROSA, 2019).

A representação dos cromossomos e a diversidade populacional também são temas

recorrentes. (BEAN, 1994) introduziu o conceito de chaves aleatórias (*Random Keys*), uma codificação que simplifica a manipulação de cromossomos e garante a viabilidade das soluções, sendo amplamente adotada em trabalhos como o de (LONDE et al., 2025). (RAFSANJANI; RIYAHI, 2020) exploram técnicas para manter a diversidade, o que é um desafio conhecido dos AGs, como discutido por (SUDHOLT, 2016) e (BLICKLE; THIELE, 1996).

Além disso, a aplicação de hiper-heurísticas e funções de *fitness* multi-objetivo tem ganhado destaque. (AKARSU; KÜÇÜKDENIZ, 2022) propõem uma hiper-heurística que usa o AG para escolher a melhor regra de despacho, adaptando-se a diferentes cenários. (SHAO; KSHITIJ; KIM, 2024) e (SHAO; KIM, 2022) expandem o JSSP para problemas mais complexos, considerando múltiplos objetivos além do makespan, como a utilização das máquinas.

Finalmente, alguns estudos se concentram em aspectos mais fundamentais dos AGs, como a adaptação e o comportamento de operadores genéticos. (BELL, 2022) investiga o uso da mutação Gaussiana para auto-adaptação, e (KATOCH; CHAUHAN; KUMAR, 2020)) fornecem uma revisão abrangente sobre a evolução dos algoritmos genéticos, reforçando sua relevância contínua na otimização de problemas complexos. Esses trabalhos, juntamente com a dissertação (JUNIOR, 2015), reforçam a importância de uma representação cromossômica adequada para o desempenho dos AGs em problemas de sequenciamento.

Tabela 3.1: *Resumo dos artigos analisados, com origem, título e contribuição principal*

ID	Referência)	Título	Origem / Base de Publicação	Comentário
1	(AKARSU; KÜÇÜKDENIZ, 2022)	Job Shop scheduling with genetic algorithm-based hyperheuristic approach	International Advanced Researches and Engineering Journal (DergiPark)	Hiper-heurística baseada em AG que seleciona heurísticas de despacho para resolver o JSSP.
2	(BELL, 2022)	Applications of Gaussian Mutation for Self Adaptation in Evolutionary Genetic Algorithms	Journal of Machine Learning in Fundamental Sciences JMLFS-ID	Analisa mutações gaussianas para autoajuste em AGs evolucionários.
3	(BORDIGNON; SANTOS; SILVA, 2022)	Estudo sobre as operações críticas no sequenciamento de tarefas em sistemas produtivos do tipo Job Shop	Research, Society and Development (Revista brasileira)	Foco nas operações críticas e seu impacto no desempenho do sequenciamento Job Shop.
4	(CONSTANTINO; SEGURA, 2021)	A parallel memetic algorithm with explicit management of diversity for the Job Shop scheduling problem	Applied Intelligence (Springer, Scopus/WoS)	AG memético paralelo com controle explícito da diversidade e Busca Local.
5	(KATOCH; CHAUHAN; KUMAR, 2020)	A review on genetic algorithm: past, present, and future	Multimedia Tools and Applications (Springer, Scopus/WoS)	Revisão abrangente do histórico, aplicações e tendências futuras dos AGs.
6	(LONDE et al., 2025)	Random-Key Genetic Algorithms: Principles and Applications	Handbook of Heuristics, 2nd edition	Explora GA com chaves aleatórias (Random-Key GA), destacando aplicações práticas.
7	(JUNIOR, 2015)	Estudo comparativo de diferentes representações cromossômicas nos algoritmos genéticos em problemas de sequenciamento da produção em Job Shop	Dissertação (UNINOVE, Brasil)	Compara representações cromossômicas no desempenho do AG para Job Shop.
8	(PALACIOS et al., 2014)	Robust swarm optimisation for fuzzy open shop scheduling	Natural Computing (Springer, Scopus/WoS)	Otimização enxame aplicada a shop scheduling com incertezas e lógica fuzzy.
9	(RAFSANJANI; RIYAHI, 2020)	A New Hybrid Genetic Algorithm for Job Shop Scheduling Problem	Computers and Industrial Engineering (Elsevier, Scopus/WoS)	AG híbrido com Busca Local e controle da diversidade para evitar convergência prematura.
10	(VIANA; JUNIOR; CONTRERAS, 2020)	A Modified Genetic Algorithm with Local Search Strategies and Multi-Crossover Operator for Job Shop Scheduling Problem	Sensors (MDPI, Scopus/WoS)	Modifica operadores de cruzamento e aplica Busca Local no AG para JSSP.
11	(SHAO; KIM, 2022)	An Adaptive Job Shop Scheduler Using Multilevel Convolutional Neural Network and Iterative Local Search	IEEE Access (IEEE, Scopus/WoS)	Usa CNN com Busca Local iterativa para adaptar sequenciamentos dinamicamente.
12	(SHAO; KSHITIJ; KIM, 2024)	GAILS: An Effective Multi-Object Job Shop Scheduler Based on Genetic Algorithm and Iterative Local Search	Scientific Reports (Nature, Scopus/WoS)	GAILS: combina AG e Busca Local para otimização multiobjetivo de JSSP e FJSSP.
13	(SUDHOLT, 2016)	Benefits of population diversity in evolutionary algorithms: A survey of rigorous runtime analyses	Artificial Intelligence (Elsevier, Scopus/WoS)	Analisa rigorosamente o impacto da diversidade em algoritmos evolutivos.
14	(ROSA, 2019)	Algoritmo Genético Híbrido Baseado na Análise de Componentes Principais do *Fitness Landscape* para o Problema de Job Shop Scheduling	Tese (UNINOVE, Brasil)	HGA com estratégias de diversificação e intensificação baseadas na análise de componentes principais do *fitness landscape*.
15	(ROSA; PEREIRA, 2024)	An intensification approach based on fitness landscape characteristics for job shop scheduling problem	Journal of Combinatorial Optimization	Abordagem de intensificação para o JSSP baseada nas características da paisagem de aptidão (*fitness landscape*) que utiliza a topologia de "grande vale"para gerenciar a diversidade da população em AGs e evitar estagnação em ótimos estritamente locais.

Capítulo 4

Metodologia

Neste capítulo, apresenta-se a metodologia adotada para a resolução do problema de *JSSP*. Serão detalhadas as etapas da proposta, desde a concepção inicial até os procedimentos de avaliação e análise dos resultados. Neste trabalho, é proposta a utilização de um AG com codificação via chaves aleatórias (RKGA)(BEAN, 1994), combinado com um decodificador Giffler–Thompson na variante *Non-Delay* (GT-ND) (MOONEN; JANSSENS, 2007). Esta abordagem é complementada por uma estratégia de Busca Local ativa, visando refinar e melhorar a qualidade final das soluções encontradas.

Em problemas de JSSP, *benchmarks* são coleções de problemas padrão amplamente utilizados pela literatura para avaliar e comparar algoritmos de otimização. Cada *benchmark* é composto por diversas instâncias, que representam casos específicos com números definidos de *jobs*, máquinas, ordens operacionais e tempos de processamento. O uso dessas instâncias padronizadas permite medir o desempenho dos métodos propostos em condições equivalentes às de outros trabalhos publicados.

As instâncias definidas por (FISHER; THOMPSON, 1963) e (LAWRENCE, 1984), são *benchmarks* padrões, usados para testar e avaliar o desempenho de algoritmos de otimização e sistemas de controle em pesquisa operacional. As instâncias que serão trabalhadas neste trabalho são definidas na Tabela 4.1 abaixo.

Tabela 4.1: *Características das Instâncias de JSSP Utilizadas*

Instância	**jobs** (n)	**Máquinas** (m)	**Tamanho** ($n \times m$)	**Melhor solução conhecida** ($C_{\max}$)
FT06	6	6	6×6	55
LA01	10	5	10×5	666
LA02	10	5	10×5	655
LA03	10	5	10×5	597
LA04	10	5	10×5	590
LA05	10	5	10×5	593
LA06	15	5	15×5	926
LA07	15	5	15×5	890
LA08	15	5	15×5	863
LA09	15	5	15×5	951
LA10	15	5	15×5	958

Fonte: Adaptado de Fisher e Thompson

Os *benchmarks* FT06, e LA01 a LA10 originalmente definidos por (FISHER; THOMPSON, 1963), e por (LAWRENCE, 1984) foram selecionados para estes testes. A seleção destas instâncias FT06, LA01 a LA10 para os testes preliminares baseia-se na sua representatividade e relevância para o problema de sequenciamento JSSP. Estes *benchmarks* são amplamente utilizados, permitindo uma comparação direta dos resultados com ou-

tras abordagens já existentes. Além disso, o conjunto FT06, LA01 a LA10 apresenta uma variedade de características que são importantes para avaliar o desempenho do algoritmo.

Os algoritmos foram implementados na linguagem Python (versão 3.135). Para a manipulação de dados e cálculos das instâncias do JSSP, foram empregadas as bibliotecas NumPy e Pandas, enquanto a interface gráfica para seleção das instâncias foi desenvolvida com a biblioteca Tkinter. A visualização dos sequenciamentos resultantes foi viabilizada por meio de gráficos de Gantt customizados, gerados com a biblioteca Matplotlib. Os resultados de cada rodada experimental foram exportados e consolidados em planilhas eletrônicas através da biblioteca OpenPyXL.

No exemplo ilustrado na Figura 4.1, mostra o processo da solução proposta como funciona com a primeira solução somente do AG e depois a solução final. Ao executar o RKGA, e obter uma resposta, o algoritmo verifica se o resultado do makespan é o melhor conhecido. Caso não seja o valor do melhor conhecido, é realizada a Busca Local que é determinada pela maior folga entre o fim de uma job em uma máquina e o início da mesma job em outra máquina, isso fica ilustrado através da seta indicada na imagem, onde a seta mostra o exato local onde será a Busca Local na solução proposta na imagem para o problema de FT06.

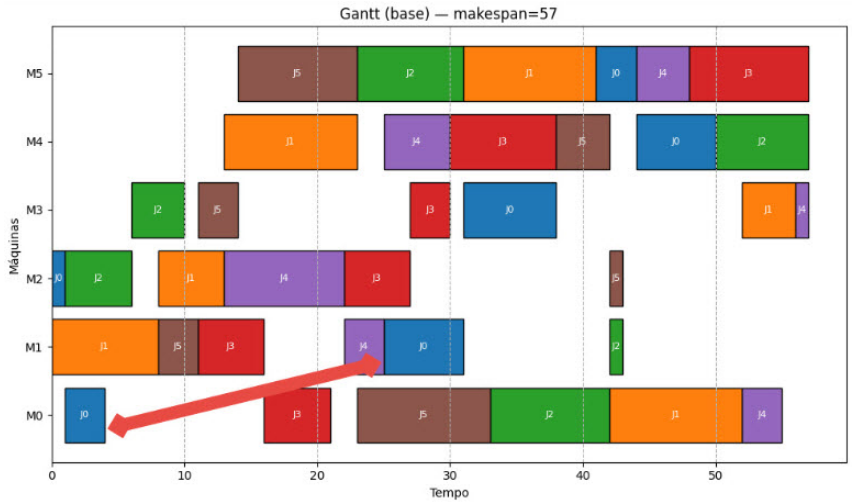


Figura 4.1: *FT06 - Subótimo - Indicação da maior folga*

Após a primeira etapa na qual não foi identificado o melhor conhecido, é realizado atrasos pontuais nas jobs selecionadas e a ordem das jobs é redistribuída, conforme ilustrado na imagem 4.2, onde é possível observar que o valor do *makespan* é outro.

A execução do algoritmo foi realizada em 10 iterações para cada *benchmark*. Os resultados obtidos em cada teste foram registrados em um arquivo .csv, formando a base de dados para a subsequente análise dos resultados.

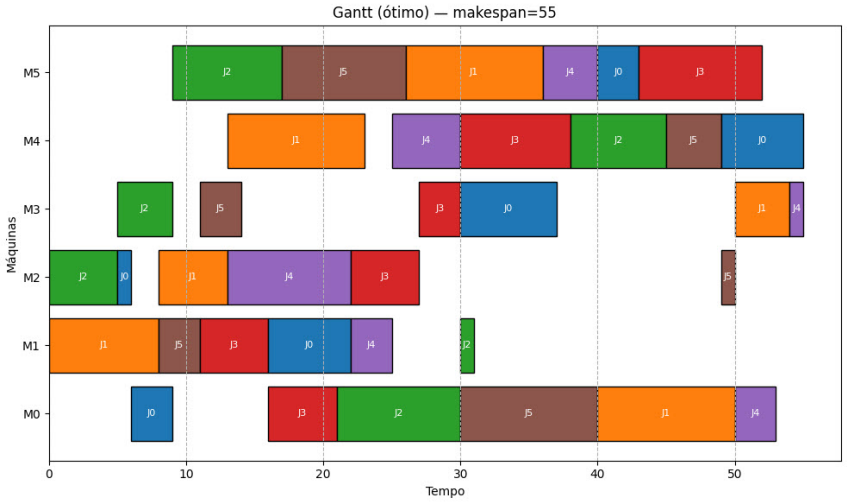


Figura 4.2: *Problema FT06: Solução ótima (makespan = 55)*

4.1 Execução do Algoritmo Genético de Chaves Aleatórias (RKGA)

A abordagem proposta neste estudo utiliza o RKGA para a busca global de soluções, seguido por uma etapa de Busca Local ativa (o refinamento) em soluções não atrasadas com o objetivo de melhorar o *makespan* das instâncias propostas.O funcionamento básico pode ser dividido nas etapas de representação e transformação descritas a seguir:

4.1.1 Codificação (*Encoding*):

A codificação em um algoritmo genético (AG) é a representação de uma solução potencial (um "cromossomo") em um formato que o algoritmo possa manipular. Dentre esta representação, alguns tópicos se fazem necessários, tais como:

1. As soluções são representadas por um vetor de n chaves aleatórias (BELL, 2022; RESENDE; RIBEIRO, 2016), denominado cromossomo (LONDE et al., 2025).

2. Uma chave aleatória (*random key*) é um número real aleatório gerado no intervalo contínuo $[0, 1)$ (BEAN, 1994; BELL, 2022; MITCHELL, 1996; RESENDE; RIBEIRO, 2016).

3. Essa representação, proposta por Bean (1994) (LONDE et al., 2025), torna o processo evolutivo do RKGA independente do problema (LONDE et al., 2025; BEAN, 1994), pois todos os operadores evolutivos podem ser especificados para os cromossomos (vetores de números reais) (LONDE et al., 2025; BEAN, 1994).

4.1.2 Decodificação (*Decoding*):

Um decodificador (*decoder*) é um algoritmo determinístico (LONDE et al., 2025) que recebe o vetor de chaves aleatórias como entrada e mapeia-o para uma solução do problema de otimização, computando o custo ou *fitness* da solução (LONDE et al., 2025; BEAN, 1994; RESENDE; RIBEIRO, 2016).

A variante *Non-Delay* do decodificador de Giffler–Thompson (MOONEN; JANSSENS, 2007) garante que nenhuma máquina permaneça ociosa se houver alguma operação disponível para ser processada. Isso é obtido selecionando, a cada passo, a operação com menor tempo de início possível entre todas as operações elegíveis, e agendando-a imediatamente nesse instante. Dessa forma, sempre que uma máquina puder iniciar uma operação, ela o faz sem atraso intencional, caracterizando um agendamento não atrasado *(Non-Delay Schedule)*

4.1.3 Inicialização da População:

A população é o conjunto de todos os indivíduos (soluções candidatas) que estão sendo testados em uma determinada geração. Se o seu algoritmo está tentando resolver um problema de JSSP, a população é um conjunto de soluções diferentes criadas para verificar qual é a melhor.

O processo começa criando uma população de `80` indivíduos (cromossomos). Cada cromossomo é uma lista de números aleatórios (as *"random-keys"*) entre 0 e 1. O tamanho dessa lista é igual ao número total de operações no problema (número de *jobs* × número de máquinas). O valor de 80 foi escolhido para a população por ser um valor que garante diversidade genética, um princípio fundamental para a exploração eficaz do espaço de busca em Algoritmos Genéticos (EIBEN; SMITH, 2015).

4.1.3.1 Estrutura do Indivíduo (Cromossomo)

A interpretação dos cromossomos é realizada por meio de ordenação: seus valores determinam a prioridade de execução das operações. Essa abordagem facilita a manipulação dos cromossomos e evita a geração de soluções inviáveis, pois, conforme destacado por (BEAN, 1994), a principal vantagem das chaves aleatórias é que, se qualquer vetor de chaves puder ser interpretado como uma solução viável, então qualquer vetor formado por cruzamento também será viável. A estrutura do cromossomo contém:

- Valores (Alelos): Cada número na sequência corresponde ao identificador do *job* (índice 0, 1, 2, etc.) ao qual pertence a operação a ser agendada.
- Comprimento: O comprimento total da sequência é igual ao número total de opera-

ções no problema ($n \times m$), onde n é o número de *jobs* e m é o número de máquinas. Para um problema 3×3, como o exemplo base (3 *jobs* $\times$ 3 operações por *job*), o comprimento da sequência é $3 \times 3 = 9$.

- Precedência: A ordem posicional dos números no vetor indica a prioridade de execução.

A Figura 4.3 mostra mostra a estrura do cromosso no momento que se é criado.

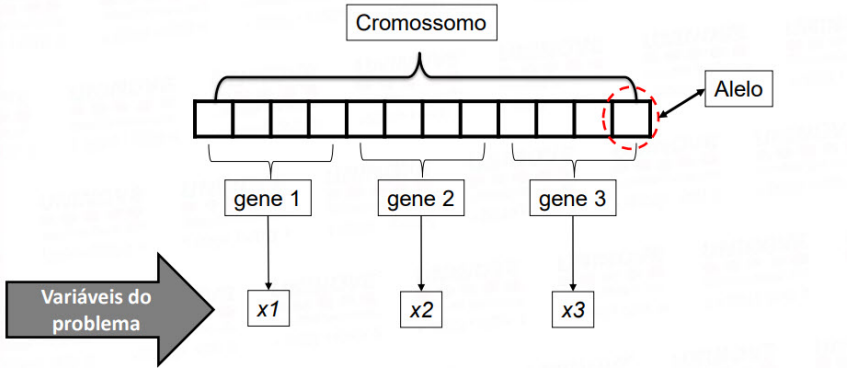


Figura 4.3: *Cromossomo*

4.1.3.2 Funcionamento da Sequência

Segundo (BEAN, 1994), diferentemente de um AG tradicional baseado em sequências explícitas de *jobs*, no RKGA o cromossomo não define diretamente a ordem de execução das operações. Em vez disso, cada gene do vetor de chaves aleatórias está associado a uma operação específica do problema, e o valor numérico da chave representa sua prioridade.

Durante o processo de decodificação, o vetor é ordenado implicitamente: sempre que há mais de uma operação elegível para execução, o decodificador utiliza os valores das chaves como critério de desempate. Assim, a sequência efetiva de operações não é lida diretamente do cromossomo, mas construída dinamicamente pelo decodificador Giffler–Thompson *(Non-Delay)* (MOONEN; JANSSENS, 2007), que:

1. respeita a precedência tecnológica de cada *job*;
2. verifica o menor tempo possível em que cada operação pode iniciar;
3. seleciona a operação com maior prioridade (menor chave aleatória) entre as elegíveis;
4. agenda essa operação imediatamente, caracterizando um sequenciamento não atrasado (*Non-Delay*).

4.1.3.3 Decodificação e Avaliação da Aptidão:

Cada cromossomo na população (que é uma sequência de chaves aleatórias) é decodificado para produzir um sequenciamento real. As chaves aleatórias são usadas como prioridades para agendar as operações, realizado pelo algoritmo de decodificação, o Decodificador Giffler–Thompson (não-atrasado) (MOONEN; JANSSENS, 2007):

- A operação com a menor chave aleatória entre as operações elegíveis (aquelas cujas predecessoras foram concluídas) é agendada primeiro, respeitando as precedências e a disponibilidade das máquinas.
- O resultado da decodificação é um sequenciamento completo, cujo tempo total é o *makespan*.

O *makespan* é a medida de aptidão (*fitness*) do cromossomo. Por se tratar de um problema de minimização, um *makespan* menor indica maior aptidão.

Para fins de ilustração, um problema 3×3 será exemplificado com uma população de 10 indivíduos, mostrando uma possível população inicial para o RKGA. Essa população é representada na Tabela 4.2 abaixo.

Tabela 4.2: *População Inicial no RKGA — Cromossomos Representados por chaves aleatórias*

Indivíduo	**Cromossomo (chaves aleatórias)**
Posição dos Genes (Operações): [O1, O2, O3, O4, O5, O6, O7, O8, O9] **jobs Correspondentes:** [J1, J1, J1, J2, J2, J2, J3, J3, J3]	
1	[0.12, 0.87, 0.44, 0.03, 0.66, 0.51, 0.29, 0.91, 0.75]
2	[0.55, 0.21, 0.98, 0.10, 0.36, 0.72, 0.84, 0.05, 0.47]
3	[0.93, 0.18, 0.57, 0.42, 0.81, 0.14, 0.23, 0.69, 0.90]
4	[0.07, 0.64, 0.32, 0.59, 0.11, 0.48, 0.95, 0.37, 0.26]
5	[0.28, 0.73, 0.15, 0.82, 0.91, 0.09, 0.33, 0.54, 0.67]
6	[0.44, 0.05, 0.71, 0.27, 0.62, 0.39, 0.86, 0.13, 0.58]
7	[0.31, 0.89, 0.17, 0.24, 0.78, 0.96, 0.41, 0.02, 0.53]
8	[0.60, 0.34, 0.12, 0.97, 0.45, 0.25, 0.19, 0.88, 0.71]
9	[0.83, 0.40, 0.08, 0.56, 0.20, 0.63, 0.47, 0.32, 0.99]
10	[0.16, 0.52, 0.69, 0.30, 0.74, 0.22, 0.90, 0.11, 0.38]

Fonte: Autor

Considere o Indivíduo 1 da Tabela 4.2, cujo cromossomo de chaves aleatórias é

$$[0.12, 0.87, 0.44, 0.03, 0.66, 0.51, 0.29, 0.91, 0.75].$$

Neste exemplo, assumimos que cada posição do vetor está associada a uma operação específica do problema 3×3, de forma que as posições 1 a 9 representam, respectivamente, as operações $O_1 = J1O1$, $O_2 = J1O2$, $O_3 = J1O3$, $O_4 = J2O1$, $O_5 = J2O2$, $O_6 = J2O3$, $O_7 = J3O1$, $O_8 = J3O2$ e $O_9 = J3O3$.

Ao ordenar as *chaves aleatórias* em ordem crescente, obtemos:

$$0.03(4) < 0.12(1) < 0.29(7) < 0.44(3) < 0.51(6) < 0.66(5) < 0.75(9) < 0.87(2) < 0.91(8),$$

onde o número entre parênteses indica a posição do gene no cromossomo. Assim, a sequência de prioridade das operações é:

$$O_4, O_1, O_7, O_3, O_6, O_5, O_9, O_2, O_8,$$

isto é,

$$J2O1,\ J1O1,\ J3O1,\ J1O3,\ J2O3,\ J2O2,\ J3O3,\ J1O2,\ J3O2.$$

Essa sequência é utilizada como critério de seleção das operações em cada etapa do agendamento. Dessa forma, garante-se que as operações de cada *job* sejam executadas na sua ordem tecnológica (O1 $\rightarrow$ O2 $\rightarrow$ O3), enquanto o RKGA explora diferentes padrões de prioridade entre operações de *jobs* distintos, o que resulta em agendamentos distintos como o mostrado imagem 4.4 do gráfico de Gantt do Indivíduo 1.

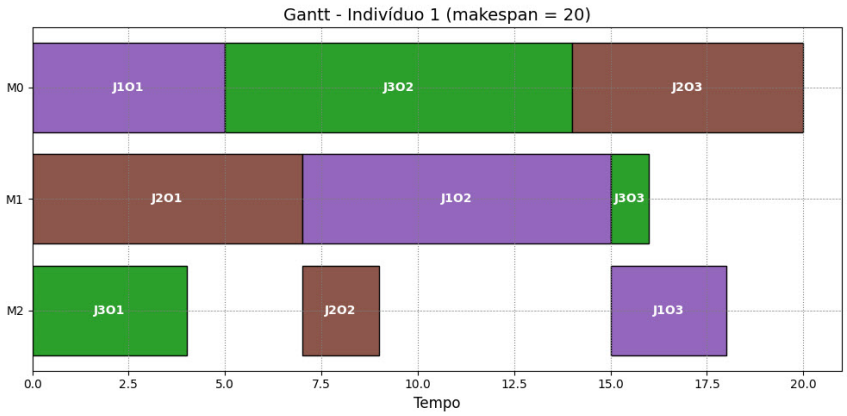


Figura 4.4: *Ilustração 3x3 - Cromossomo 1*

4.1.3.4 Exemplo de avaliação de Aptidão para uma instância 3x3

A aptidão (*fitness*) de um indivíduo no RKGA é calculada a partir do tempo total de conclusão do sequenciamento, isto é, o *makespan* $C_{\max}$ resultante da decodificação do cromossomo pelo algoritmo de Giffler–Thompson (modo *Non-Delay).* Nesta etapa, cada vetor de chaves aleatórias é convertido em um agendamento factível do problema JSSP, e o valor de aptidão é dado pelo inverso do makespan:

$$\text{fitness}(s) = \frac{1}{\text{makespan}(s)}. \tag{4.1}$$

A Tabela 4.3 apresenta um exemplo de avaliação de aptidão para a população inicial de 10 indivíduos mostrada anteriormente (Tabela 4.2), todos decodificados para a mesma instância 3×3. Cada cromossomo gera um agendamento distinto após a aplicação do decodificador Giffler–Thompson *Non-Delay.* Como consequência, os makespans obtidos variam entre os indivíduos, refletindo diferentes níveis de qualidade de solução.

Tabela 4.3: *Avaliação de Aptidão da População Inicial (Instância* 3×3 *com RKGA)*

Indivíduo	**Cromossomo (chaves aleatórias)**	**Makespan**	**Fitness** ($1/C_{\max}$)
1	[0.12, 0.87, 0.44, 0.03, 0.66, 0.51, 0.29, 0.91, 0.75]	20	0.05000
2	[0.91, 0.72, 0.88, 0.63, 0.95, 0.41, 0.33, 0.27, 0.12]	23	0.04347
3	[0.80, 0.90, 0.12, 0.77, 0.66, 0.54, 0.48, 0.39, 0.25]	25	0.04000
4	[0.93, 0.55, 0.60, 0.88, 0.79, 0.32, 0.14, 0.67, 0.29]	27	0.03704
5	[0.99, 0.74, 0.81, 0.46, 0.35, 0.92, 0.57, 0.68, 0.15]	22	0.04545
6	[0.73, 0.69, 0.91, 0.82, 0.19, 0.14, 0.35, 0.88, 0.47]	26	0.03846
7	[0.84, 0.61, 0.93, 0.57, 0.48, 0.22, 0.95, 0.33, 0.10]	24	0.04167
8	[0.67, 0.58, 0.83, 0.91, 0.72, 0.18, 0.29, 0.44, 0.37]	28	0.03571
9	[0.52, 0.95, 0.71, 0.38, 0.59, 0.87, 0.92, 0.40, 0.21]	29	0.03448
10	[0.88, 0.79, 0.56, 0.99, 0.61, 0.93, 0.47, 0.32, 0.05]	30	0.03333

Nota: Este exemplo didático apresenta cromossomos que geram makespans distintos após a decodificação Non-Delay, permitindo observar a variação de aptidão entre os indivíduos da população inicial.

4.1.3.5 Loop Evolutivo (Gerações):

O algoritmo executa por um número predefinido de gerações, estipulado em `120` ciclos completos de seleção, cruzamento e mutação. A adoção de um número fixo de gerações como critério de parada é uma prática consolidada na literatura de Algoritmos Genéticos para impor um limite computacional seguro à busca (EIBEN; SMITH, 2015). A escolha específica deste valor visa balancear a qualidade da solução final (convergência) com o tempo de execução exigido, uma vez que um número excessivamente grande de gerações demanda recursos massivos sem necessariamente garantir melhorias proporcionais no sequenciamento (SHAO; KSHITIJ; KIM, 2024). Além disso, o limite adequado de ciclos evolutivos depende intrinsecamente da complexidade do problema tratado, sendo que,

em muitos cenários de otimização, em torno de uma centena de iterações já se mostra suficiente para atingir uma boa convergência de forma eficiente (WANG; ZHANG, 2020).

Dentro de cada geração, seguindo os processos definidos por (EIBEN; SMITH, 2015) (LONDE et al., 2025), os seguintes passos acontecem:

- **Elitismo:**
  Em cada geração, uma fração de 5% dos melhores indivíduos da população é preservada sem modificações. Como a população é composta por 80 cromossomos, isso corresponde a 4 indivíduos de elite, aqueles que apresentam os menores valores de *makespan*. Esses cromossomos são copiados diretamente para a nova geração, garantindo que as melhores soluções encontradas não sejam perdidas.

  De acordo com (EIBEN; SMITH, 2015) o melhor indivíduo de cada geração, é copiado diretamente para a próxima, assegurando que soluções superiores não sejam perdidas durante as recombinações e mutações, ou seja, a solução com o menor makespan encontrada na geração atual é automaticamente preservada e não corre o risco de ser perdida durante os processos de cruzamento e mutação, e que por acaso, pode gerar soluções piores Essa estratégia é utilizada em algoritmos Random-Key, e colabora para a convergência eficiente sem comprometer aptidões já encontradas (LONDE et al., 2025).

  O indivíduo de elite da geração atual é aquele com o menor *makespan* e, consequentemente, maior aptidão. No exemplo considerado, esse papel é desempenhado pelo Indivíduo 1, que possui $C_{\max} = 20$ e fitness = 0,05000, conforme resumido na Tabela 4.4. Esse indivíduo é preservado por elitismo e incluído diretamente na nova geração, garantindo que sua qualidade não seja perdida por operadores genéticos.

Tabela 4.4: *Indivíduo de Elite da Geração Atual*

Indivíduo	**Makespan**	**Aptidão**
1	20	0.05000

Fonte: Autor.

- Seleção por Torneio: Segundo (EIBEN; SMITH, 2015), Um grupo de 2 indivíduos é escolhido aleatoriamente para formar um "torneio", pois tamanho 2 é uma escolha comum e eficaz que oferece um bom balanço entre a pressão seletiva e a manutenção da diversidade. O indivíduo com melhor aptidão é selecionado como pai, repetindo-se até formar o número desejado de pais. Essa abordagem mantém o equilíbrio entre exploração e preservação de diversidade, enquanto garante a pressão seletiva (LONDE et al., 2025).

Com as aptidões avaliadas para os indivíduos da população, inicia-se o processo de seleção para gerar os pais da próxima geração. A Tabela 4.5 apresenta um exemplo de seleção por torneio binário (tamanho 2), utilizando os valores de aptidão da população inicial mostrada nas Tabelas 4.2 e 4.3. O Numero de pais depende do tamanho da população, como é um torneio binário, para uma população de tamanho 10 teremos 5 pais.

Tabela 4.5: *Torneios Binários — Seleção de Pais*

Torneio	**Indivíduos Sorteados**	**Aptidões**	**Vencedor**
1	5 vs 3	0.04545 vs 0.04000	5
2	1 vs 7	0.05000 vs 0.04167	1
3	2 vs 6	0.04347 vs 0.03846	2
4	4 vs 10	0.03704 vs 0.03333	4
5	8 vs 9	0.03571 vs 0.03448	8

Fonte: Autor.

- **Seleção:**
  Os demais 76 indivíduos da nova população são gerados por meio de seleção por torneio binário. Nessa abordagem, dois cromossomos são escolhidos aleatoriamente da população atual, e o indivíduo com o menor *makespan* (melhor aptidão) vence o torneio, sendo selecionado como um dos pais para a etapa de reprodução. Esse processo é repetido continuamente até que haja pais suficientes para gerar toda a nova população.

  Com base nos torneios apresentados na Tabela 4.5, que estão servindo como base para ilustração, os indivíduos 5, 1, 2, 4 e 8 foram selecionados como pais para a próxima fase (cruzamento). Esse processo é repetido conforme a quantidade de pais necessária. A Figura 4.5 ilustra, de forma esquemática, o processo de seleção por torneio combinado com elitismo.

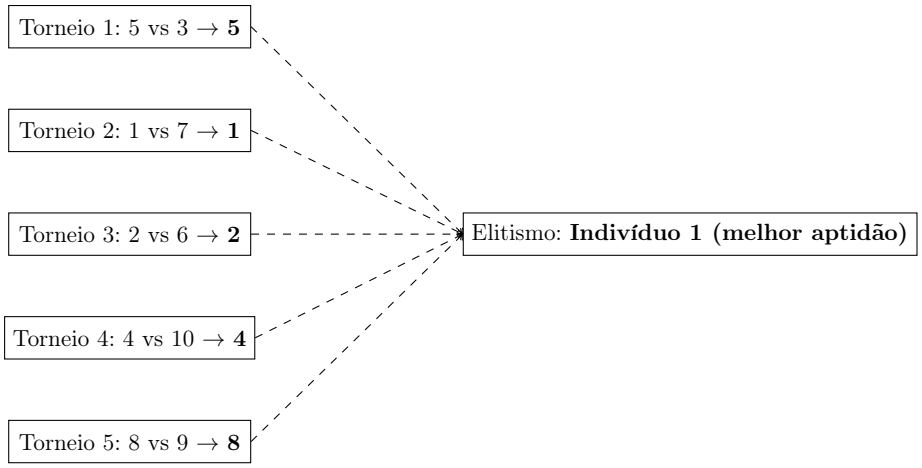


Figura 4.5: *Seleção por Torneio e Elitismo — Exemplo 3×3*

Fonte: Autor.

- **Cruzamento (Crossover):**
  Com uma taxa de cruzamento de 80%, pares de pais selecionados trocam partes de seus cromossomos. O algoritmo utiliza o cruzamento de ponto único, no qual cada par de pais gera dois novos cromossomos filhos. A taxa de 80% segue recomendações frequentes na literatura, que sugerem valores elevados para estimular a recombinação e aumentar a exploração do espaço de busca.

  Uma alta taxa de recombinação é fundamental para gerar novas combinações de genes, prevenindo a estagnação da população e promovendo a busca por soluções otimizadas, enquanto se mantém um equilíbrio entre exploração e explotação. De acordo com (KATOCH; CHAUHAN; KUMAR, 2020), valores elevados de taxa de cruzamento (como 80%) são comuns em algoritmos genéticos, e a ideia central também se reflete no esquema RKGA proposto neste trabalho.

  Basicamente o cruzamento funciona com as seguintes etapas:

- uma fração p_e dos melhores indivíduos é copiada diretamente para a próxima geração (elite);
- uma fração p_m de indivíduos é gerada aleatoriamente (mutantes);
- o restante da população é preenchido por descendentes produzidos via cruzamento enviesado entre um pai da elite e um pai não-elite.

A Tabela 4.6 a seguir ilustra, de forma didática, a ideia de favorecer indivíduos com maior aptidão na participação do cruzamento, selecionando, neste exemplo, os oito indivíduos com melhor *fitness* para atuarem como candidatos a pais são os classificados.

Tabela 4.6: *Seleção por Aptidão para Cruzamento (Taxa de 80%) —Exemplo com População de 10*

Ordem	**ID Original**	**Cromossomo (chaves aleátorias)**	**Aptidão**	**Participa do Cruzamento?**
1	1	[0.12, 0.87, 0.44, 0.03, 0.66, 0.51, 0.29, 0.91, 0.75]	0.05000	Sim
2	5	[0.28, 0.73, 0.15, 0.82, 0.91, 0.09, 0.33, 0.54, 0.67]	0.04545	Sim
3	2	[0.91, 0.72, 0.88, 0.63, 0.95, 0.41, 0.33, 0.27, 0.12]	0.04347	Sim
4	7	[0.31, 0.89, 0.17, 0.24, 0.78, 0.96, 0.41, 0.02, 0.53]	0.04167	Sim
5	3	[0.93, 0.18, 0.57, 0.42, 0.81, 0.14, 0.23, 0.69, 0.90]	0.04000	Sim
6	6	[0.73, 0.69, 0.91, 0.82, 0.19, 0.14, 0.35, 0.88, 0.47]	0.03846	Sim
7	4	[0.93, 0.55, 0.60, 0.88, 0.79, 0.32, 0.14, 0.67, 0.29]	0.03704	Sim
8	8	[0.67, 0.58, 0.83, 0.91, 0.72, 0.18, 0.29, 0.44, 0.37]	0.03571	Sim
9	9	[0.52, 0.95, 0.71, 0.38, 0.59, 0.87, 0.92, 0.40, 0.21]	0.03448	**Não**
10	10	[0.88, 0.79, 0.56, 0.99, 0.61, 0.93, 0.47, 0.32, 0.05]	0.03333	**Não**

Nota: A tabela está ordenada pela Aptidão (do maior para o menor). Os oito indivíduos de maior aptidão (80% da população) foram selecionados para participar do cruzamento. Os indivíduos 9 e 10 (com as menores aptidões) foram excluídos desta etapa.

- **Mutação:**
  Após o cruzamento, cada gene (chave aleatória) dos cromossomos filhos pode sofrer uma mutação gaussiana com probabilidade de 2% (BELL, 2022). Quando ocorre, o

valor da chave é perturbado pela soma de um ruído aleatório normalmente distribuído, mantido dentro do intervalo $[0, 1]$. Esse processo introduz diversidade adicional e ajuda a evitar a convergência prematura.

A mutação é um operador fundamental classificado como operador de variação unário (*unary variation operator*), aplicado a um único cromossomo para produzir um descendente modificado (EIBEN; SMITH, 2015). Este operador é responsável por introduzir alterações aleatórias (*random changes*) na estrutura do cromossomo (BORDIGNON; SANTOS; SILVA, 2022).

O principal papel da mutação é manter a diversidade genética da população (LONDE et al., 2025; BORDIGNON; SANTOS; SILVA, 2022). Ela atua como o motor do processo evolutivo, promovendo a diversificação da busca *diversification of search* (LONDE et al., 2025).

A mutação é crucial porque:

- Opõe-se à Convergência: O operador opõe-se à convergência e substitui o material genético que pode ser perdido durante os processos de cruzamento (*crossover*) e reprodução (GONÇALVES; MENDES; RESENDE, 2005).
- Prevenção de Ótimos estritamente Locais: Ao introduzir variabilidade, a mutação evita que o algoritmo fique preso em ótimos estritamente locais (*local optima*) (JúNIOR, 2021; LONDE et al., 2025; GONÇALVES; MENDES; RESENDE, 2005).
- Garantia de Acessibilidade: A mutação assegura que a probabilidade de se alcançar qualquer ponto do espaço de busca nunca será zero (GOLDBERG, 1989).

O mecanismo da mutação depende da forma de codificação utilizada. Segundo (LONDE et al., 2025) no contexto do RKGA, o operador de mutação é tipicamente realizado pela injeção de mutantes diretamente na nova população, onde os indivíduos mutantes são vetores de chaves aleatórias (*random keys*) gerados de forma uniformemente aleatória no intervalo contínuo $[0, 1)$. Em representações de valores reais, como as chaves aleatórias, é frequentemente utilizada a Mutação Gaussiana (*Gaussian Mutation*) (SHYLO et al., 2021; EIBEN; SMITH, 2015). Essa técnica envolve a adição de um valor retirado de uma distribuição Gaussiana ao gene, o que é um método comum em algoritmos genéticos com codificação de parâmetros reais (SHYLO et al., 2021)

A taxa de mutação é a probabilidade com que a mutação ocorrerá (MITCHELL, 1996; WANG; ZHANG, 2020). Geralmente, ela deve ser mantida baixa (LONDE et al., 2025), pois uma taxa muito alta de mutação torna a busca essencialmente aleatória (*random search*) (JúNIOR, 2021).

- **Atualização da População:**
  A nova população é composta pelos indivíduos de elite preservados e pelos filhos gerados por seleção, cruzamento e mutação. Após sua formação, essa população substitui integralmente a população anterior, permitindo o avanço do algoritmo para a próxima geração.

A Tabela 4.7 resume os valores adotados e suas respectivas origens.

Tabela 4.7: *Parâmetros adotados para a execução do RKGA com Busca Local*

Parâmetro	**Valor**	**Justificativa / Fonte**
Tamanho da População ($\|\mathcal{P}\|$)	80	Literatura (WANG; ZHANG, 2020; KATOCH; CHAUHAN; KUMAR, 2020)
Número de Gerações (G)	120	Literatura (WANG; ZHANG, 2020; SANTOS; PEREIRA; ALMEIDA, 2025)
Taxa de Cruzamento (pc)	80%	Literatura (WANG; ZHANG, 2020)
Taxa de Mutação Gaussiana (pm)	2%	Literatura (BELL, 2022)
Fração de Elitismo (pe)	5%	Literatura (BEAN, 1994; LONDE et al., 2025)
Decodificador	GT-ND	Literatura (MOONEN; JANSSENS, 2007)
Incremento de atraso ($delay$)	5 u.t.	Empírico (Testes preliminares)

O Autor.

4.1.4 Diagrama fluxo RKGA

A Figura 4.6 mostra, passo a passo, como o RKGA evolui uma população ao longo de uma geração. O diagrama destaca como o algoritmo avalia os indivíduos, preserva o melhor deles (elitismo), seleciona pais por torneio, realiza cruzamentos e mutações, e finalmente constrói a próxima população. Esse fluxo ajuda a visualizar o funcionamento geral do método através de uma demonstração de um problema 3x3 com uma população inicial de 10 cromossomos.

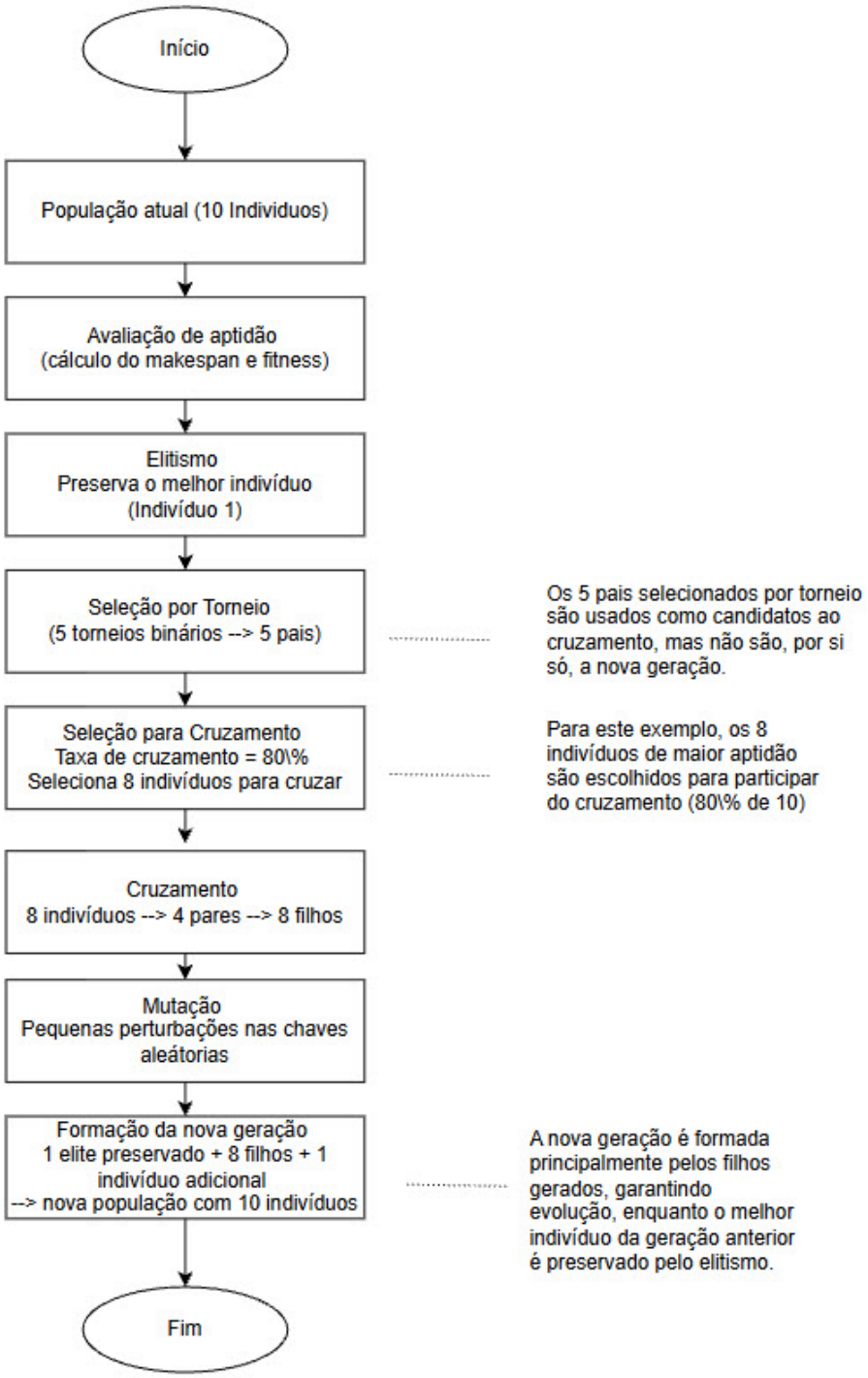


Figura 4.6: *Fluxo do ciclo evolutivo com seleção por torneio, elitismo e cruzamento (exemplo com população de 10 indivíduos).*

Fonte: Autor

4.2 Fluxo de Processamento do Algoritmo Genético de chaves aleatórias (RKGA)

Considerando o entendimento dos principais componentes do Algoritmo Genético, incluindo a codificação chaves aleatórias, a avaliação de aptidão e os operadores genéticos (seleção, cruzamento e mutação), é fundamental apresentar a sequência em que esses processos interagem.

A Figura 4.7 ilustra o ciclo de vida completo do algoritmo proposto neste trabalho, destacando as condições de parada e o momento em que a melhor solução conhecida é retornada.

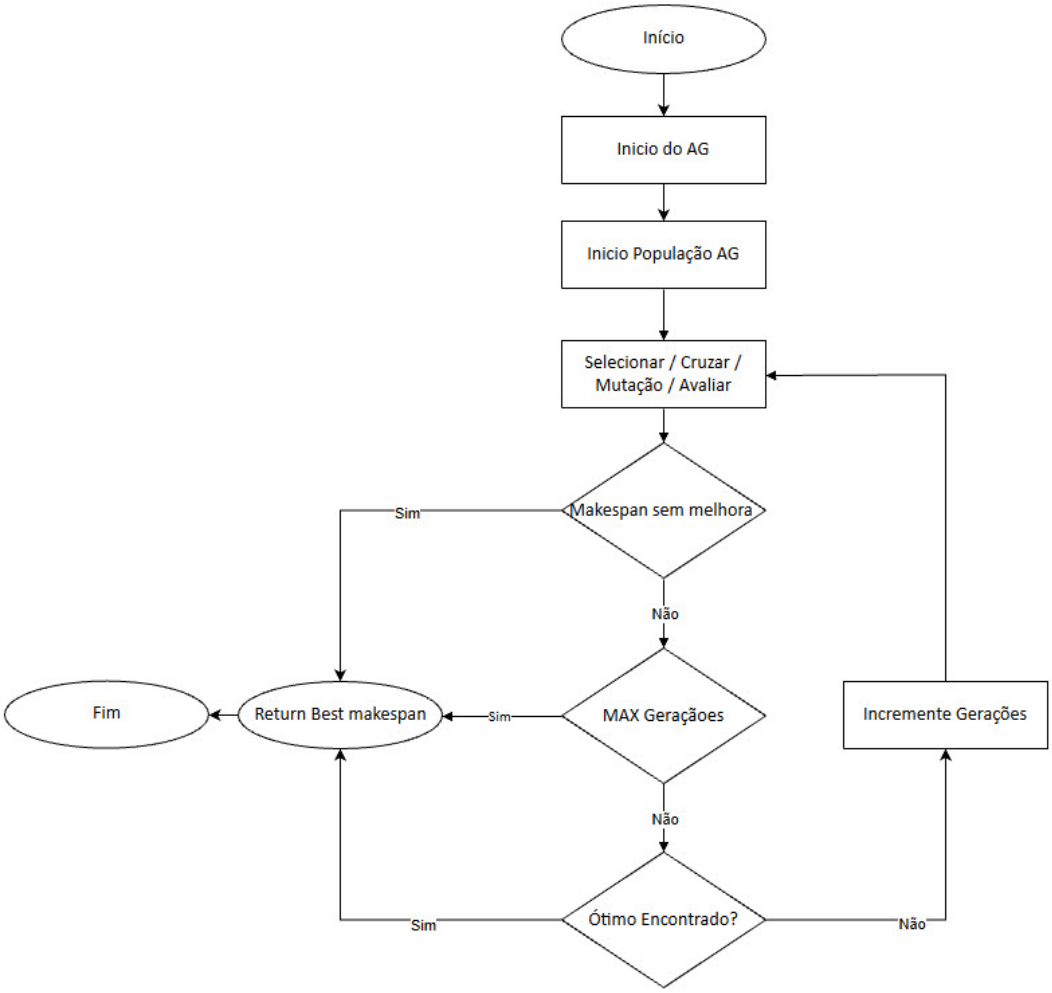


Figura 4.7: *Fluxograma RKGA*

Fonte: Autor

4.3 Fase de Refinamento Local (ou Pós-Processamento)

Após a execução do algoritmo genético (RKGA) na etapa inicial, que busca uma solução global para o *JSSP*, o critério de aperfeiçoamento propõe aplicar uma etapa adicional de refinamento, através da Busca Local, caso o *makespan* encontrado não seja o melhor conhecido. O objetivo dessa fase é melhorar o *makespan* da solução obtida, explorando ajustes que o RKGA, pode não identificar. Essa estratégia complementa a busca global, permitindo explorar regiões promissoras do espaço de busca que exigem manipulações mais específicas na estrutura do sequenciamento.

O refinamento recebe o nome de busca ativa porque, ao introduzir atrasos controlados em operações específicas, o algoritmo deixa o espaço estrito das soluções não atrasadas produzidas pelo decodificador Giffler–Thompson *(Non-Delay)*. Embora toda solução não atrasada seja ativa, o inverso não é verdadeiro: ao permitir atrasos estratégicos, o refinamento passa a explorar o espaço mais amplo das soluções ativas, onde operações podem ser deslocadas desde que não existam antecipações possíveis. Esse deslocamento abre novas possibilidades de reorganização do sequenciamento que não seriam alcançadas se apenas soluções *Non-Delay* fossem utilizadas.

O processo inicia com a identificação das folgas (*slacks*) entre operações consecutivas de cada *job*, que pode ser verificado através do exemplo da Figura 4.8. A folga representa o intervalo em que uma operação subsequente ainda não precisa começar para respeitar a ordem tecnológica. Assim, trata-se do período pelo qual a operação anterior pode ser atrasada sem violar a precedência da *job*.

Operações com grandes folgas são bons pontos para refinamento, pois ao atrasar levemente uma operação que possui folga disponível, provoca-se um deslocamento controlado no sequenciamento que pode permitir que operações de outras *jobs* (especialmente aquelas em disputa pela mesma máquina) sejam adiantadas durante a nova decodificação. Desse modo, a melhoria no *makespan* não ocorre pelo atraso em si, mas pela reorganização resultante do sequenciamento quando conflitos de máquina são alterados.

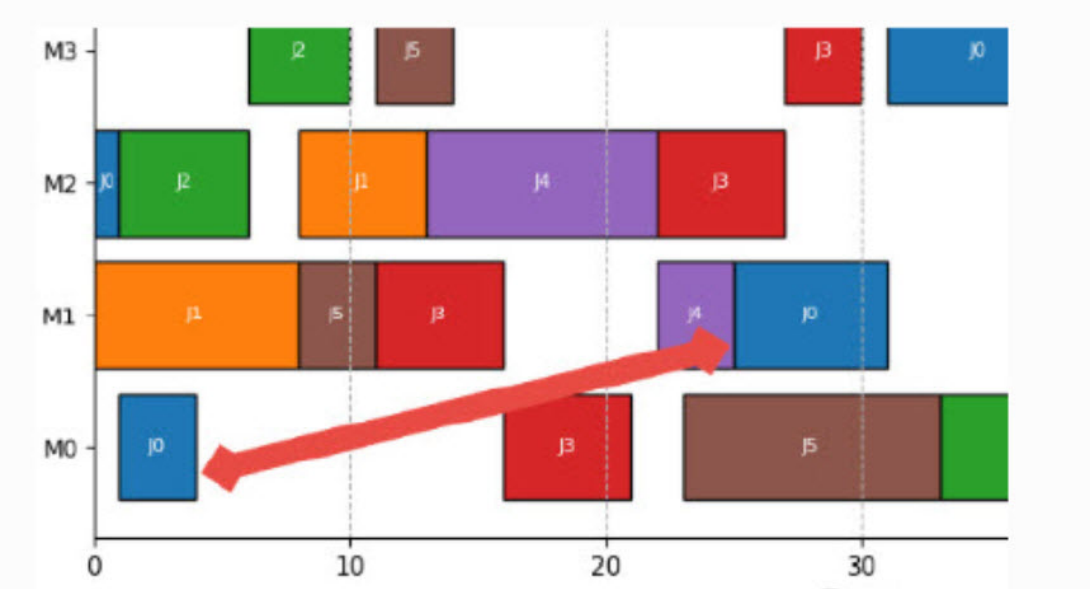


Figura 4.8: *Maior folga encontrada.*

Fonte: Autor.

4.3.1 Critério de Parada e Dinâmica de Reexecução do RKGA no Refinamento

O algoritmo proposto é estruturado em duas etapas interdependentes: uma busca global baseada no *Random-Key Genetic Algorithm* (RKGA) e uma fase de refinamento local via Busca Local Ativa. O fluxo operacional integra o critério de parada à detecção de melhores conhecidos, conforme detalhado a seguir.

Inicialmente, o RKGA é executado com uma população $|\mathcal{P}| = 80$ ao longo de $G = 120$ gerações. Nesta fase, cada indivíduo é decodificado pelo algoritmo de Giffler–Thompson em modo *Non-Delay* (GT-ND) (MOONEN; JANSSENS, 2007), gerando um *makespan* inicial $C_{\max}^{(0)}$. O algoritmo finaliza esta etapa retornando o melhor cromossomo (**key**) da população inicial.

Para garantir a eficácia do método e equilibrar o custo computacional com a qualidade das soluções, os parâmetros do algoritmo foram ajustados com base em recomendações consolidadas da literatura (detalhadas na revisão bibliográfica) e em testes empíricos preliminares.

Caso o melhor valor conhecido ($C_{\max}^{\star}$) seja fornecido e a condição $C_{\max}^{(0)} \leq C_{\max}^{\star}$ seja satisfeita, a execução é encerrada imediatamente. Esta estratégia de *early stop* evita o custo computacional desnecessário da busca local quando a solução ideal já foi alcançada pelo AG puro.

Se o melhor conhecido não for atingido ($C_{\max}^{(0)} > C_{\max}^{\star}$), inicia-se o refinamento local. O algoritmo identifica as folgas (*slacks*) entre operações consecutivas de cada *job*, priorizando aquelas com maiores intervalos ociosos para a aplicação de atrasos estratégicos (**s**).

Diferente de buscas locais tradicionais que alteram o cromossomo, esta etapa mantém as prioridades (*random keys*) fixas e manipula o vetor de atrasos. Para cada incremento testado (definido empiricamente em passos de 5 unidades de tempo no código), o RKGA é reexecutado integralmente ($G = 120$ gerações). Esta reexecução busca encontrar a melhor combinação de chaves aleatórias sob a nova restrição temporal imposta pelo vetor $\mathbf{s}'$:

$$\mathbf{key}^{\star}(\mathbf{s}') = \arg\min_{\mathbf{key}} \; C_{\max}\big(\text{GT-ND}(\mathbf{key}, \mathbf{s}')\big)$$

É importante ressaltar que a inserção de atrasos estratégicos (*delays*) não altera diretamente a estrutura ou os genes (chaves aleatórias) do cromossomo original. No RKGA adotado, o cromossomo codifica exclusivamente as prioridades de agendamento das operações. O atraso atua, em vez disso, como um parâmetro externo (uma restrição ou condição inicial) imposto ao ambiente do decodificador.

O refinamento é, portanto, um processo iterativo que se repete enquanto houver melhoria no *makespan*. Para cada incremento de atraso testado em uma operação com folga, o algoritmo genético principal é reexecutado integralmente (por mais 120 gerações) sob esse novo cenário restritivo.

Ao ser reexecutado, o RKGA evolui uma nova população e naturalmente descobre uma nova configuração ótima de chaves aleatórias adaptada àquele atraso. Se essa nova execução produzir um agendamento com um resultado superior (menor *makespan*), o sistema atualiza a melhor solução global. Essa atualização não consiste em modificar matematicamente o cromossomo anterior, mas sim em substituí-lo pelo novo melhor cromossomo recém-descoberto pela reexecução do AG, associando-o à configuração de atraso que permitiu essa melhoria.

O processo encerra-se sob duas condições:

1. Otimização Total: Quando $C_{\max}(\mathbf{s}') \leq C^{\star}_{\max}$ é atingido durante qualquer reexecução do RKGA.

2. Estagnação: Quando todos os candidatos de atraso em todas as folgas disponíveis (da maior para a menor) são testados sem que ocorra redução do melhor *makespan* registrado.

Esta técnica de "permutação induzida" é eficaz com a codificação *random-key*, pois pequenas alterações nos tempos de liberação dos *jobs* forçam o decodificador a explorar variações estruturais no espaço de soluções ativas que seriam inacessíveis a um decodificador *Non-Delay* convencional.

A Figura 4.9 mostra todas as etapas desse processo: identificar as folgas, escolher qual operação ajustar, aplicar pequenos atrasos, reexecutar o AG e verificar se houve melhoria. O refinamento continua até que não seja possível melhorar mais a solução ou até que o limite de iterações seja atingido.

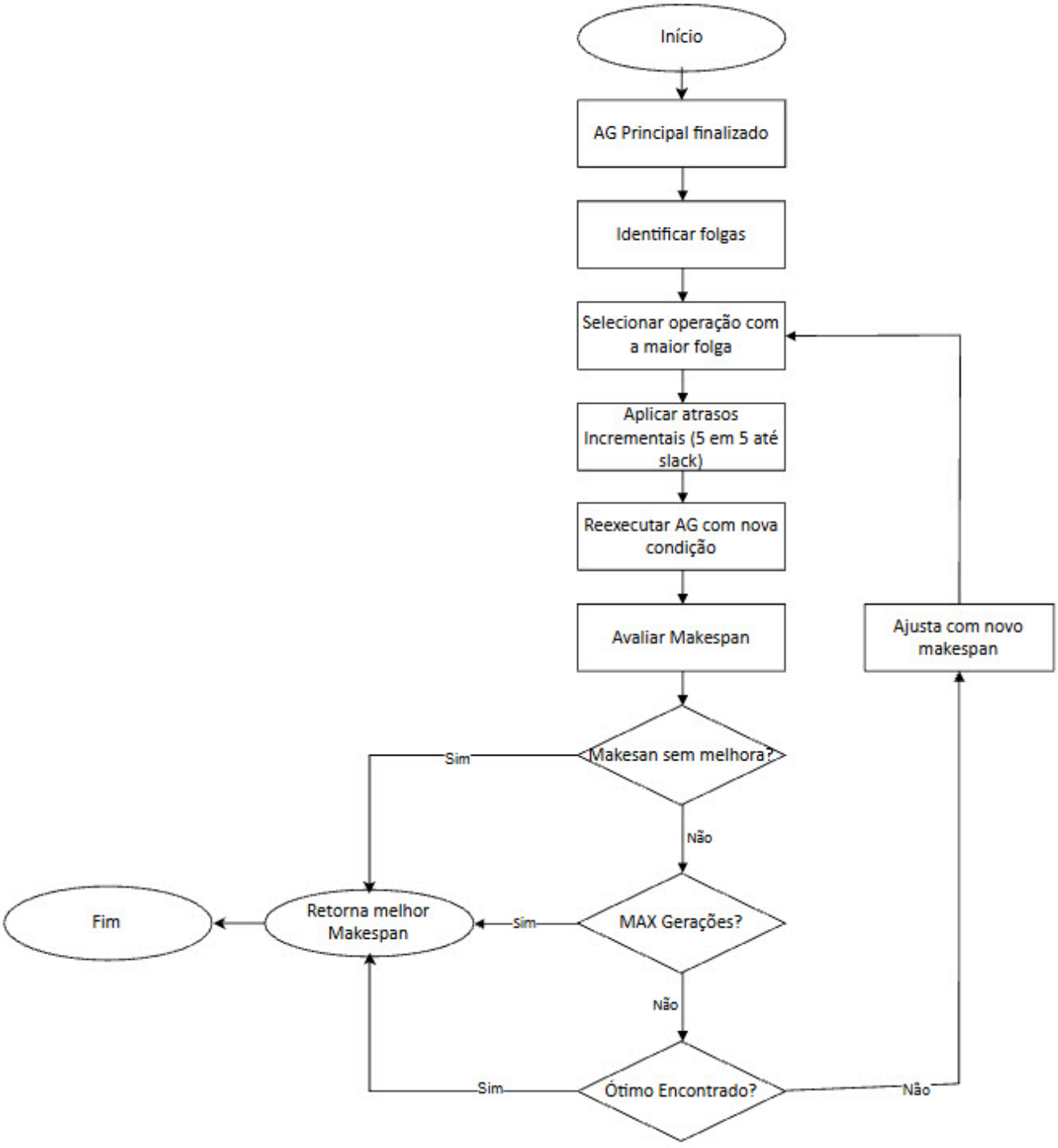


Figura 4.9: *Fluxograma AG - Refinamento por Busca Local*

Fonte: Autor

4.4 Identificação de folga e permutação da *job*

A estratégia de busca local proposta opera de forma híbrida, mantendo a estrutura genética do RKGA inalterada enquanto refina os tempos de entrada das tarefas no sistema. Diferentemente dos operadores genéticos que alteram as prioridades (chaves aleatórias), esta etapa mantém o cromossomo fixo e explora a introdução de *atrasos estratégicos* (*delay times*). O objetivo é explorar as folgas (*slacks*) operacionais existentes no sequenciamento para escapar de ótimos estritamente locais (GONÇALVES; MENDES; RESENDE, 2005).

O refinamento ocorre pela manipulação do vetor de atrasos iniciais por *job*, definido como:

$$\mathbf{s} = [s_1, s_2, \ldots, s_n],$$

onde s_j representa um tempo de espera forçado imposto ao início do processamento do *job* j.

A aplicação deste atraso s_j não apenas posterga o início de uma tarefa, mas induz uma permutação indireta das operações nas máquinas. Ao adicionar um atraso $s_j > 0$, altera-se o tempo de disponibilidade (r_k) das operações daquele *job* no algoritmo de Giffler–Thompson (MOONEN; JANSSENS, 2007). Isso pode modificar a composição do Conjunto de Conflito em uma iteração crítica: uma operação que anteriormente seria escolhida por ter alta prioridade pode, devido ao atraso imposto, tornar-se indisponível naquele instante. Consequentemente, a máquina é liberada para processar outra tarefa.

Essa mecânica permite preencher a ociosidade no sequenciamento. Se uma máquina crítica possui uma folga antes de processar uma *job* prioritário, o atraso estratégico pode forçar esse *job* a ceder sua vez, permitindo que operações menores ocupem essa folga, compactando o *makespan* final.

A busca local explora a vizinhança definida pela variação discreta desses atrasos. Em cada iteração, para cada *job* j, testam-se sistematicamente todos os valores de atraso d pertencentes a um conjunto finito, por exemplo, $d \in \{0, 1, 2, 3\}$. Para cada movimento candidato, constrói-se um vetor temporário $\mathbf{s}'$ tal que:

$$s'_j = d \quad \text{e} \quad s'_{j'} = s_{j'} \;\; \forall j' \neq j. \tag{4.2}$$

A cada novo vetor $\mathbf{s}'$ gerado, o decodificador GT (Giffler–Thompson) é reexecutado integralmente. Isso caracteriza uma nova simulação completa do sistema, pois a alteração no tempo de início de um único *job* reverbera por toda a cadeia de restrições de precedência, exigindo que todos os tempos de início e término sejam recalculados para determinar o novo *makespan* $C_{\max}(\mathbf{s}')$.

O novo sequenciamento não é uma combinação planejada, ele aparece como uma consequência dos atrasos temporais inseridos no sistema.

4.5 Ambiente de Execução

Os experimentos foram realizados em um computador com as seguintes especificações:

- Processador: Intel Core i7, 2.7 GHz;
- Memória RAM: 32 GB;
- Sistema operacional: Windows 11;
- Linguagem de programação: Python 3.13.5;
- Bibliotecas utilizadas: NumPy, Matplotlib, Tkinter.
- Software: Visual Studio 1.99.3

4.6 Métricas Avaliadas

As principais métricas utilizadas para análise dos resultados foram:

- **Makespan**: tempo total necessário para concluir todas as jobs;
- **Tempo de execução**: tempo computacional necessário para obter a melhor solução em cada execução.

Capítulo 5

Resultados

Esta seção apresenta os resultados obtidos a partir da implementação da abordagem proposta para o JSSP. Os experimentos computacionais foram conduzidos utilizando os *benchmarks* da literatura e os dados a seguir ilustram as primeiras observações sobre a eficácia e o desempenho da abordagem desenvolvida.

A redução do *makespan* observada nas séries de testes realizados demonstra uma diminuição real no tempo total necessário para concluir todas as tarefas no ambiente simulado. Se aplicarmos essa otimização em uma fábrica real que opera com ciclos de produção semelhantes, os ganhos obtidos, que chegaram a superar 5% de melhoria no tempo total após o refinamento (como verificado na instância LA04), podem significar uma redução direta no tempo de espera entre os processos. Na prática industrial, um sequenciamento adequado que minimiza o *makespan* reflete-se em um menor acúmulo de materiais em processamento (*Work-In-Process* - WIP), o que reduz significativamente os custos de armazenamento e de oportunidade (SILVA et al., 2012). Além disso, essa otimização conduz a um aumento tangível da produtividade, uma vez que a melhor utilização da capacidade das máquinas permite que mais lotes sejam concluídos no mesmo intervalo de tempo (GONÇALVES; MENDES; RESENDE, 2005; BORDIGNON; SANTOS; SILVA, 2022).

Para se ter um impacto direto nos custos operacionais em empresas que operam com o JSSP, melhorar o *makespan* é essencial. Por exemplo, se uma linha de produção consegue completar os mesmos produtos em menos tempo, os gastos com energia elétrica, mão de obra e manutenção tendem a diminuir.I

5.1 Comparação de Instâncias e Melhoria do *Makespan*

A Tabela 5.1 apresenta os resultados consolidados em um formato compacto, focando na performance, no tempo de execução e no percentual de atingimento do melhor conhecido.

Tabela 5.1: *Resultados Consolidados e Otimizados: Makespan, Tempo e Atingimento do melhor conhecido*

Inst.	Tam.	M. conhecido (M.K.)	M. Inicial (Faixa)	M. Final (Melhor)	Tempo Médio (s)	Melhoria Máx. (%)	(% M. conhecido)
ft06	6 × 6	55	57-57	55	8.57	3.51	100%
la01	10 × 5	666	666-666	666	2.12	0.00	100%
la02	10 × 5	655	662-677	655	1037.93	3.11	20%
la03	10 × 5	597	619-624	606	1225.35	2.88	0%
la04	10 × 5	590	611-623	590	1217.19	5.30	100%
la05	10 × 5	593	593-593	593	2.09	0.00	100%
la06	15 × 5	926	926-926	926	4.25	0.00	100%
la07	15 × 5	890	890-890	890	4.90	0.00	100%
la08	15 × 5	863	863-863	863	3.85	0.00	100%
la09	20 × 5	951	951-951	951	4.55	0.00	100%
la10	20 × 5	958	958-958	958	4.67	0.00	100%

Os dados apresentados nas Tabelas 5.2, 5.3 e 5.4 ajudam a ilustrar a eficácia da proposta deste trabalho, onde é possível verificar qual foi o *makespan* inicial, encontrado através do AG com RKGA, e o tempo final da execução. Em casos onde o *makespan* inicial não foi o melhor encontrado e os tempos superiores a 0 mostra a porcentagem de melhoria após aplicar o refinamento.

Tabela 5.2: *Resultados das instâncias FT06 até LA03*

Instância	Rodada	Makespan Inicial	Makespan Final	Tempo (s)	Melhoria (%)	Chegou m. conhecido
ft06	1	57	55	7.42	3.51	sim
ft06	2	57	55	7.87	3.51	sim
ft06	3	57	55	7.79	3.51	sim
ft06	4	57	55	7.97	3.51	sim
ft06	5	57	55	9.42	3.51	sim
ft06	6	57	55	10.49	3.51	sim
ft06	7	57	55	11.23	3.51	sim
ft06	8	57	55	7.30	3.51	sim
ft06	9	57	55	7.97	3.51	sim
ft06	10	57	55	8.23	3.51	sim
la01	1	666	666	2.22	0.00	sim
la01	2	666	666	2.22	0.00	sim
la01	3	666	666	2.05	0.00	sim
la01	4	666	666	2.21	0.00	sim
la01	5	666	666	2.07	0.00	sim
la01	6	666	666	2.07	0.00	sim
la01	7	666	666	2.11	0.00	sim
la01	8	666	666	2.06	0.00	sim
la01	9	666	666	2.10	0.00	sim
la01	10	666	666	2.11	0.00	sim
la02	1	670	658	1809.65	1.79	não
la02	2	677	658	1704.93	2.81	não
la02	3	670	658	1614.68	1.79	não
la02	4	662	658	428.17	0.60	não
la02	5	668	658	495.21	1.50	não
la02	6	676	655	555.84	3.11	sim
la02	7	668	658	548.80	1.50	não
la02	8	670	661	1510.21	1.34	não
la02	9	672	658	1060.94	2.08	não
la02	10	672	655	650.87	2.53	sim
la03	1	624	606	783.56	2.88	não
la03	2	624	606	786.07	2.88	não
la03	3	624	606	1051.23	2.88	não
la03	4	619	606	900.52	2.10	não
la03	5	619	606	906.54	2.10	não
la03	6	619	606	906.55	2.10	não
la03	7	619	606	2230.24	2.10	não
la03	8	619	606	1069.97	2.10	não
la03	9	623	606	2079.99	2.73	não
la03	10	623	606	1538.83	2.73	não

Tabela 5.3: *Resultados das instâncias LA04 até LA07 (Continuação)*

Instância	Rodada	Makespan Inicial	Makespan Final	Tempo (s)	Melhoria (%)	Chegou m. conhecido
la04	1	623	590	1906.49	5.30	sim
la04	2	611	590	706.49	3.44	sim
la04	3	611	590	701.39	3.44	sim
la04	4	611	590	1906.50	3.44	sim
la04	5	611	590	701.39	3.44	sim
la04	6	611	590	701.39	3.44	sim
la04	7	611	590	727.31	3.44	sim
la04	8	611	590	727.31	3.44	sim
la04	9	611	590	2046.83	3.44	sim
la04	10	611	590	2046.83	3.44	sim
la05	1	593	593	2.05	0.00	sim
la05	2	593	593	2.05	0.00	sim
la05	3	593	593	2.04	0.00	sim
la05	4	593	593	2.03	0.00	sim
la05	5	593	593	2.03	0.00	sim
la05	6	593	593	2.03	0.00	sim
la05	7	593	593	2.13	0.00	sim
la05	8	593	593	2.15	0.00	sim
la05	9	593	593	2.21	0.00	sim
la05	10	593	593	2.22	0.00	sim
la06	1	926	926	4.22	0.00	sim
la06	2	926	926	4.22	0.00	sim
la06	3	926	926	4.23	0.00	sim
la06	4	926	926	4.17	0.00	sim
la06	5	926	926	4.22	0.00	sim
la06	6	926	926	4.39	0.00	sim
la06	7	926	926	4.41	0.00	sim
la06	8	926	926	4.19	0.00	sim
la06	9	926	926	4.19	0.00	sim
la06	10	926	926	4.20	0.00	sim
la07	1	890	890	4.07	0.00	sim
la07	2	890	890	4.08	0.00	sim
la07	3	890	890	3.98	0.00	sim
la07	4	890	890	4.04	0.00	sim
la07	5	890	890	4.23	0.00	sim
la07	6	890	890	4.24	0.00	sim
la07	7	890	890	3.93	0.00	sim
la07	8	890	890	4.25	0.00	sim
la07	9	890	890	7.84	0.00	sim
la07	10	890	890	8.35	0.00	sim

Tabela 5.4: *Resultados das instâncias LA08 até LA10 (Continuação)*

Instância	Rodada	Makespan Inicial	Makespan Final	Tempo (s)	Melhoria (%)	Chegou m. conhecido
la08	1	863	863	1.88	0.00	sim
la08	2	863	863	1.94	0.00	sim
la08	3	863	863	4.41	0.00	sim
la08	4	863	863	4.39	0.00	sim
la08	5	863	863	4.35	0.00	sim
la08	6	863	863	4.32	0.00	sim
la08	7	863	863	4.32	0.00	sim
la08	8	863	863	4.31	0.00	sim
la08	9	863	863	4.32	0.00	sim
la08	10	863	863	4.31	0.00	sim
la09	1	951	951	4.20	0.00	sim
la09	2	951	951	4.24	0.00	sim
la09	3	951	951	4.37	0.00	sim
la09	4	951	951	4.44	0.00	sim
la09	5	951	951	4.39	0.07	sim
la09	6	951	951	4.41	0.07	sim
la09	7	951	951	4.23	0.07	sim
la09	8	951	951	4.25	0.07	sim
la09	9	951	951	6.64	0.07	sim
la09	10	951	951	4.33	0.07	sim
la10	1	958	958	4.52	0.00	sim
la10	2	958	958	4.51	0.00	sim
la10	3	958	958	4.67	0.00	sim
la10	4	958	958	4.71	0.00	sim
la10	5	958	958	4.69	0.00	sim
la10	6	958	958	4.71	0.00	sim
la10	7	958	958	4.72	0.00	sim
la10	8	958	958	4.71	0.00	sim
la10	9	958	958	4.72	0.00	sim
la10	10	958	958	4.71	0.00	sim

5.2 Análise dos resultados

Os resultados obtidos para as instâncias FT06 e LA01-LA10, detalhados nas Tabelas 5.1, 5.2, 5.3 e 5.4, mostram algumas percepções sobre a capacidade do algoritmo em otimizar o *makespan*. Uma análise mais aprofundada dos dados revela os seguintes pontos:

5.2.1 Análise do *Makespan* (Inicial e Final) e Percentual de melhor conhecido

O algoritmo demonstrou consistentemente sua capacidade de refinar soluções iniciais aleatórias, que em muitos casos apresentavam um *makespan* superior ao *makespan* inicial. A melhoria percentual máxima, que variou de 0.00% até 5.30% (na instância LA04), ilustra a eficácia da abordagem em encontrar soluções melhores, especialmente em cenários onde a solução inicial estava distante do melhor conhecido.

É notável que para instâncias como FT06, LA01, LA04, LA05, LA06, LA07, LA08, LA09 e LA10, o algoritmo atingiu o *makespan* melhor conhecido em 100% das rodadas (AO).

Por outro lado, as instâncias LA02 (*Makespan* Final 655, Melhor conhecido 655) e LA03 (*Makespan* Final 606, Melhor Conhecido 597) apresentaram os maiores desafios: - LA02 atingiu o *makespan* melhor em apenas 20% das rodadas. - LA03 não atingiu o *makespan* melhor conhecido (0%) em nenhuma rodada, parando no valor de 606.

5.2.2 Desempenho e Tempo de Execução

O tempo de execução é um fator crítico para a aplicabilidade de um algoritmo em ambientes reais. Conforme detalhado na Tabela 5.1, o Tempo Médio (s) variou significativamente entre as instâncias:

Instâncias que Convergiram Rapidamente (FT06, LA01, LA05-LA10): Para a maioria das instâncias que atingiram o melhor conhecido em 100% das rodadas (FT06, LA01, LA05 a LA10), o tempo de execução foi extremamente baixo, variando de 2.09 a 8.57 segundos em média. Isso indica que o algoritmo é altamente eficiente para problemas de complexidade média e maior (até LA10) quando a convergência é rápida.

O tempo de execução aumentou drasticamente para as instâncias mais difíceis, como LA02 e LA03. O tempo médio de processamento foi de 1037.93 segundos (aproximadamente 17.3 minutos) para LA02 e 1225.35 segundos (aproximadamente 20.4 minutos) para LA03. Este aumento é esperado, pois o algoritmo gasta mais tempo explorando o espaço de busca na tentativa de encontrar o *makespan* melhor conhecido.

5.2.3 Comparação de Desempenho com Trabalho Similar

No trabalho de (ROSA, 2019), embora não contenha dados detalhados de tempos de execução, pois segundo a autora, o custo computacional (tempo de processamento) não foi o foco principal da análise naquela etapa, priorizando a eficácia da hibridização, podemos fazer uma comparação referente aos resultados. Ambos os trabalhos utilizam abordagens híbridas fundamentadas em algoritmos genéticos para minimizar o *makespan* no problema de *(JSSP)*. A principal diferença reside na técnica de hibridização: o trabalho de (ROSA, 2019) utiliza a Análise de Componentes Principais (PCA) para gerenciar a diversidade da população de forma *online* e um algoritmo genético binário bidimensional para intensificação. Já este trabalho utiliza o Algoritmo Genético de Chaves Aleatórias (RKGA) seguido por um refinamento de Busca Local ativa baseado na inserção estratégica de atrasos (*delays*) em operações com folgas selecionadas.

Os resultados demonstram que:

- Instâncias de Convergência Rápida: Em problemas considerados "fáceis"ou de menor dimensão, como FT06, LA01 e as instâncias de LA05 a LA10, ambas as abordagens atingiram o valor melhor conhecido (BKS) de forma consistente .

- Desempenho em Instâncias Difíceis:

  – Na instância LA03, a abordagem de (ROSA, 2019) obteve um resultado ligeiramente superior, alcançando um *makespan* de 603, enquanto a proposta deste trabalho parou em 606.

    A análise do *fitness landscape* (paisagem de aptidão) ajuda a explicar a topologia do espaço de busca do JSSP. O trabalho de (ROSA; PEREIRA, 2024) identificou que o *landscape* do problema possui uma topologia de "grande vale"(*big valley*), o que significa que os melhores ótimos locais estão concentrados e agrupados em torno do ótimo global. A heurística híbrida proposta neste trabalho navega de forma eficiente nessa topologia: o RKGA (operando no espaço não atrasado) realiza uma varredura ampla do ”vale” em busca de regiões promissoras, enquanto a Busca Local ativa, por meio da inserção de atrasos nas folgas (*slacks*), atua como uma descida refinada até o fundo do vale (o ótimo).

    Considerando a dinâmica das classes de escalonamento, é possível formular a hipótese de que as melhores soluções locais distribuídas no *big valley* sejam estritamente não atrasadas (*non-delay*), enquanto o ótimo global possa ser uma solução puramente ativa (localizada fora do espaço das soluções não atrasadas). Se essa premissa for verdadeira, a abordagem proposta apresenta, em tese, um alto potencial de desempenho nesse contexto, desde que o RKGA encontre uma solução inicial robusta dentro desse *big valley* para que a busca local a refine.

Entretanto, a confirmação definitiva dessa hipótese está fora do escopo deste trabalho, requerendo investigações estatísticas mais aprofundadas em trabalhos futuros.

Por outro lado, o "engessamento"do *landscape* em certas instâncias, como a LA03, é explicado por Bordignon, Santos e Silva (BORDIGNON; SANTOS; SILVA, 2022). O espaço de busca dessa instância é extremamente acidentado devido à forte correlação positiva entre os tempos médios de processamento e a concentração de operações críticas em apenas duas máquinas (M1 e M2). Isso cria uma paisagem repleta de mínimos locais profundos, dificultando a fuga do algoritmo e justificando a estagnação (como o valor 606 alcançado).

- Na instância LA04, ambos os métodos foram eficazes em encontrar o valor melhor conhecido de 590.
- Na instância LA02, ambos atingiram o melhor conhecido de 655, embora a abordagem deste trabalho tenha registrado esse valor em apenas 20% das rodadas, indicando um desafio maior de estabilidade para este problema específico.

Abaixo, os dados estão consolidados:

Tabela 5.5: *Comparação de Makespan: Rosa (2019) vs. Betoni (2026)*

Instância	**BKS (Melhor Conhecido)**	**Rosa (HGA)**	**Betoni (RKGA)**
FT06	55	55	55
LA01	666	666	666
LA02	655	655	655
LA03	597	**603**	606
LA04	590	590	590
LA05	593	593	593
LA06	926	926	926
LA07	890	890	890
LA08	863	863	863
LA09	951	951	951
LA10	958	958	958

Os dois métodos são altamente competitivos e eficazes para o conjunto de instâncias Lawrence de pequeno e médio porte. O HGA de (ROSA, 2019) demonstrou uma capacidade de busca ligeiramente mais refinada no caso específico da LA03. No entanto, a simplicidade do refinamento por atrasos estratégicos apresentado neste trabalho, provou ser igualmente capaz de atingir o melhor conhecido na maioria dos *benchmarks* testados com tempos de execução baixos.

Capítulo 6

Considerações finais

Este trabalho propôs uma abordagem híbrida para a resolução do problema de *Job Shop Scheduling*, combinando algoritmos genéticos com chaves aleatórias (Random-Key Genetic Algorithm) e um refinamento por Busca Local baseado em atrasos estratégicos. Os testes realizados com instâncias clássicas da literatura, como FT06, LA01 - LA10, demonstraram que a inserção controlada de atrasos em operações não críticas permitiu obter *makespans* inferiores aos gerados pelo RKGA sem Busca Local. A técnica se mostrou eficaz e com custo computacional viável.

A principal contribuição desta dissertação reside na sugestão de uma estratégia que equilibra a eficiência da busca em espaços reduzidos com a garantia de exploração de soluções de alta qualidade. Conforme discutido ao longo deste trabalho, a utilização da variante *Non-Delay* (ND) do algoritmo de *Giffler–Thompson* permite uma redução significativa do espaço de busca (MOONEN; JANSSENS, 2007). Entretanto, essa abordagem impõe o risco de excluir a solução ótima, que muitas vezes reside no espaço mais amplo das soluções ativas.

A fase de refinamento local proposta atua justamente como um mecanismo de mitigação desse risco. Ao identificar a maior folga (*slack*) e inserir atrasos incrementais de forma estratégica, o algoritmo consegue transitar do espaço *Non-Delay* para o espaço ativo de maneira controlada. Diferente de uma busca exaustiva, a proposta foca no impacto que o atraso da primeira operação de cada *job* causa no sequenciamento global, permitindo que o sistema reavalie conflitos de recursos e identifique melhorias no *makespan* que seriam inacessíveis em uma abordagem puramente *Non-Delay*.

Os resultados obtidos em instâncias clássicas demonstraram que, embora o atraso incremental de cinco unidades de tempo seja uma escolha empírica eficaz, ele abre precedentes para investigações futuras sobre o cálculo de um valor de atraso ideal, baseado na análise dinâmica das colisões em máquinas específicas.

A partir das evidências e limitações observadas nesta pesquisa, seguem abaixo as seguintes oportunidades para trabalhos futuros:

- Cálculo Dinâmico de Atrasos: Desenvolver um modelo matemático que identifique o valor exato do *delay* necessário para inverter a prioridade de operações em máquinas críticas, substituindo o incremento fixo por uma abordagem analítica.

- Análise de Escalabilidade: Aplicar a técnica de refinamento em instâncias de maior porte e complexidade, avaliando o impacto no tempo de processamento conforme o número de *jobs* e máquinas cresce.

- Hibridização de Meta-heurísticas: Comparar o desempenho da busca local ativa integrada ao RKGA com outras combinações, como *Local Search (ILS)*, *Simulated*

Annealing (SA), *Particle Swarm Optimization (PSO)*,*BKRGA Biased Random-Key Genetic Algorithm* para verificar a robustez do método frente a diferentes técnicas de exploração.

- Otimização Multiobjetivo: Expandir a lógica de refinamento para considerar indicadores além do *makespan*, como o número de tarefas, atraso máximo, etc.

Referências Bibliográficas

AKARSU, C. H.; KÜÇÜKDENIZ, T. Job shop scheduling with genetic algorithm-based hyperheuristic approach. *International Advanced Researches and Engineering Journal*, Dergipark, v. 6, n. 1, p. 16–25, 2022. Citado na pág. 34, 35.

BARAK, S.; JAVANMARD, S.; MOGHDANI, R. Dual resource constrained flexible job shop scheduling with sequence-dependent setup time. *Expert Systems*, v. 41, n. 10, p. e13669, 2024. Citado na pág. 21, 22, 23.

BEAN, J. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, INFORMS, v. 6, n. 2, p. 154–160, 1994. Citado na pág. 25, 34, 36, 38, 39, 40, 49.

BEASLEY, J. E. *An Overview of Genetic Algorithms: Part 1, Fundamentals*. [S.l.], 1993. Citado na pág. 27.

BELL, O. Applications of gaussian mutation for self adaptation in evolutionary genetic algorithms. *Journal of Machine Learning in Fundamental Sciences JMLFS-ID*, 2022. ArXiv preprint, <https://arxiv.org/abs/2201.00285>. Citado na pág. 34, 35, 38, 47, 49.

BLICKLE, T.; THIELE, L. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, MIT Press, v. 4, n. 4, p. 361–394, 1996. Citado na pág. 16, 25, 34.

BORDIGNON, J. F.; SANTOS, L. C. C.; SILVA, M. F. d. S. da. Estudo sobre as operações críticas no agendamento de tarefas em sistemas produtivos do tipo job shop. *Research, Society and Development*, v. 11, n. 5, p. e17111528035, 2022. Citado na pág. 35, 48, 59, 66.

BOUKEDROUN, M.; DUVIVIER, D.; CADI, A. A. el; POIRRIEZ, V.; ABBAS, M. A hybrid genetic algorithm for stochastic job-shop scheduling problems. *RAIRO-Operations Research*, v. 57, p. 1617–1645, 2023. Citado na pág. 25.

BäCK, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. [S.l.]: Oxford University Press, 1996. Citado na pág. 27.

CONSTANTINO, O. H.; SEGURA, C. A parallel memetic algorithm with explicit management of diversity for the job shop scheduling problem. *Applied Intelligence*, v. 51, p. 5799–5816, 2021. Citado na pág. 25, 33, 35.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. 4. ed. Rio de Janeiro: LTC, 2022. Citado na pág. 16.

EIBEN, A. E.; SMITH, J. E. *Introduction to Evolutionary Computing*. 2nd. ed. [S.l.]: Springer, 2015. Citado na pág. 16, 26, 27, 39, 43, 44, 48.

FISHER, H.; THOMPSON, G. L. Probabilistic learning combinations of local job-shop scheduling rules. In: MUTH, J. F.; THOMPSON, G. L. (Ed.). *Industrial Scheduling*. Englewood Cliffs, NJ: Prentice-Hall, 1963. p. 225–251. Citado na pág. 24, 36, 74.

GAO, L.; LI, X.; WEN, X.; LU, C.; WEN, F. A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem. *Computers & Industrial Engineering*, Elsevier, 2015. Disponível em: <http://dx.doi.org/10.1016/j.cie.2015.08.002>. Citado na pág. 21.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. [S.l.]: Addison-Wesley, 1989. Citado na pág. 26, 27, 48.

GONÇALVES, J.; MENDES, J.; RESENDE, M. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, Elsevier, v. 167, n. 1, p. 77–95, 2005. Citado na pág. 24, 25, 28, 33, 48, 57, 59.

HOLLAND, J. H. Genetic algorithms. *Scientific American*, Scientific American, a division of Nature America, Inc., v. 267, n. 1, p. 66–73, July 1992. Citado na pág. 16, 26.

JÚNIOR, L. C. dos S.; ROSA, A. de F. C.; PEREIRA, F. H. An event-based parameterized active scheduler for classical job shop problem. In: SAEED, K.; CHAKI, R.; JANEV, V. (Ed.). *Computer Information Systems and Industrial Management - 18th International Conference, CISIM 2019, Belgrade, Serbia, September 19-21, 2019, Proceedings*. [S.l.]: Springer, 2019. (Lecture Notes in Computer Science, v. 11703), p. 421–432. Citado na pág. 24, 25.

JUNIOR, V. M. *Estudo comparativo de diferentes representações cromossômicas nos algoritmos genéticos em problemas de sequenciamento da produção em job shop*. Dissertação (Dissertação de Mestrado) — Universidade Nove de Julho - UNINOVE, Programa de Pós-Graduação em Engenharia de Produção, São Paulo, 2015. Citado na pág. 25, 34, 35.

JúNIOR, J. C. A. *Modelo de arranjo linear e desigualdades válidas para o problema do job shop com minimização do makespan*. Dissertação (Dissertação de Mestrado) — Universidade Federal do Ceará, Fortaleza, 2021. Citado na pág. 25, 26, 48.

KATOCH, S.; CHAUHAN, S. S.; KUMAR, V. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, Springer US, v. 79, n. 35-36, p. 26859–26881, 2020. Citado na pág. 16, 26, 27, 34, 35, 46, 49.

LAWRENCE, S. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques*. [S.l.], 1984. Citado na pág. 36, 74.

LONDE, M. A.; PESSOA, L. S.; ANDRADE, C. E.; GONçALVES, J. F.; RESENDE, M. G. C. Random-key genetic algorithms: Principles and applications. In: MARTI, R.; PARDALOS, P.; RESENDE, M. (Ed.). *Handbook of Heuristics, 2nd edition*. New York: Springer-Nature, 2025. To appear; Date: May 30, 2025. Disponível em: <https://arxiv.org/abs/2506.02120>. Citado na pág. 25, 26, 27, 34, 35, 38, 39, 44, 48, 49.

MITCHELL, M. *An Introduction to Genetic Algorithms*. [S.l.]: MIT Press, 1996. Citado na pág. 26, 27, 38, 48.

MOONEN, M.; JANSSENS, G. K. A giffler-thompson genetic algorithm for the static job-shop scheduling problem. *Journal of Information and Computational Science*, v. 4, n. 2, p. 629–642, 2007. Citado na pág. 24, 28, 30, 36, 39, 40, 41, 49, 54, 57, 68.

MORALES, S. G.; RONCONI, D. P. Formulações matemáticas e estratégias de resolução para o problema job shop clássico. *Production*, v. 26, n. 3, p. 614–625, 2016. Citado na pág. 21, 22, 23.

PALACIOS, J.; GONZALEZ-RODRIGUEZ, I.; VELA, C.; PUENTE, J. Robust swarm optimisation for fuzzy open shop scheduling. *Natural Computing*, v. 13, n. 2, p. 145–156, 2014. Citado na pág. 33, 35.

PETROVIC, D.; CASTRO, E.; PETROVIC, S.; KAPAMARA, T. Radiotherapy scheduling. In: UYAR, A.; OZCAN, E.; URQUHART, N. (Ed.). *Automated Scheduling and Planning. Studies in Computational Intelligence*. [S.l.]: Springer, 2013. v. 505, p. 155–189. Citado na pág. 33.

PINEDO, M. L. *Scheduling: Theory, Algorithms, and Systems*. [S.l.]: Springer, 2016. Citado na pág. 16, 19, 24, 25, 28.

PULIN, I. C. *03 - Algoritmos Genéticos - Como ocorre a Evolução no Algoritmo Genético?* 2021. Disponível em: <https://www.youtube.com/watch?v=Z88i6nlWbg4>. Acesso em: 7 jan. 2026. Citado na pág. 28.

RAFSANJANI, M. K.; RIYAHI, M. A new hybrid genetic algorithm for job shop scheduling problem. *Computers and Industrial Engineering*, v. 149, p. 106849, 2020. Citado na pág. 25, 34, 35.

RESENDE, M. G. C.; RIBEIRO, C. C. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. [S.l.]: Springer, 2016. Citado na pág. 38, 39.

ROSA, A. d. F. C. *Algoritmo Genético Híbrido Baseado na Análise de Componentes Principais do Fitness Landscape para o Problema de Job Shop Scheduling*. Tese (Tese de Doutorado) — Universidade Nove de Julho - UNINOVE, Programa de Pós-Graduação em Informática e Gestão do Conhecimento, São Paulo, 2019. Citado na pág. 16, 20, 33, 35, 65, 67.

ROSA, A. d. F. C.; PEREIRA, F. H. An intensification approach based on fitness landscape characteristics for job shop scheduling problem. *Journal of Combinatorial Optimization*, v. 47, p. 77, 2024. Citado na pág. 35, 65.

SANTOS, R.; PEREIRA, J.; ALMEIDA, A. An automated parallel genetic algorithm with parametric adaptation and termination mechanisms. *Scientific Reports*, 2025. Aplica critério adaptativo + máximo de gerações. Citado na pág. 49.

SHAO, X.; KIM, C. S. An adaptive job shop scheduler using multilevel convolutional neural network and iterative local search. *IEEE Access*, v. 10, p. 88079–88092, 2022. Citado na pág. 34, 35.

SHAO, X.; KSHITIJ, F. S.; KIM, C. S. Gails: An effective multi-object job shop scheduler based on genetic algorithm and iterative local search. *Scientific Reports*, v. 14, p. 2068, 2024. Disponível em: <https://www.nature.com/articles/s41598-024-51778-1>. Citado na pág. 34, 35, 43.

SHYLO, V. P.; GLYBOVETS, M. M.; GULAYEVA, N. M.; NIKISHCHIKHINA, K. V. Genetic algorithm of tournament crowding based on gaussian mutation. *Cybernetics and Systems Analysis*, v. 56, p. 231–242, 2020. Citado na pág. 27.

SHYLO, V. P.; GLYBOVETS, M. M.; GULAYEVA, N. M.; NIKISHCHIKHINA, K. V. Application of gauss mutation genetic algorithm to optimize neural network parameters. *Mathematical Problems in Engineering*, 2021. Citado na pág. 48.

SILVA, E. B. da; COSTA, M. G.; SILVA, M. F. de Souza da; PEREIRA, F. H. Avaliação de regras de sequenciamento da produção em ambientes job shop e flow shop por meio de simulação computacional. *Exacta*, v. 10, n. 1, p. 70–81, 2012. Citado na pág. 24, 25, 59.

SUDHOLT, D. Benefits of population diversity in evolutionary algorithms: A survey of rigorous runtime analyses. *Artificial Intelligence*, Elsevier, v. 233, p. 84–110, 2016. Citado na pág. 25, 27, 34, 35.

SöRENSEN, K.; GLOVER, F. Metaheuristics. In: _____. [S.l.: s.n.], 2013. p. 960–970. ISBN 978-1-4419-1137-7. Citado na pág. 16.

VIANA, M. S.; JUNIOR, O. M.; CONTRERAS, R. C. A modified genetic algorithm with local search strategies and multi-crossover operator for job shop scheduling problem. *Sensors*, v. 20, n. 18, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/18/5440>. Citado na pág. 33, 35.

WANG, L.; ZHANG, M. Choosing mutation and crossover ratios for genetic algorithms. *Information*, v. 10, n. 12, p. 390, 2020. Citado na pág. 44, 48, 49.

YAMADA, T. *Studies on metaheuristics for jobshop and flowshop scheduling problems*. 133 p. Tese (Doutorado) — Kyoto University, Kyoto, Japan, nov 2003. Disponível em: <http://www.kecl.ntt.co.jp/as/members/yamada/YamadaThesis.pdf>. Acesso em: 27-08-2015. Disponível em: <http://www.kecl.ntt.co.jp/as/members/yamada/YamadaThesis.pdf>. Citado na pág. 20.

Apêndice A

Instâncias de *benchmark* Utilizadas

A.1 Benchmarks

Os *benchmarks* FT06 elaborado por (FISHER; THOMPSON, 1963) e os *benchmarks* LA01 - LA10 elaborados por (LAWRENCE, 1984) foram utilizados no trabalho. A seguir seguem as tabelas com os detalhes de cada um deles.

Tabela A.1: *Instância FT06 — 6 jobs, 6 máquinas (Melhor conhecido: 55)*

job	Op1	Op2	Op3	Op4	Op5	Op6
0	M1/1	M2/3	M3/6	M4/7	M5/3	M0/6
1	M2/8	M3/5	M4/10	M5/10	M0/10	M1/4
2	M3/5	M4/4	M5/8	M0/9	M1/1	M2/7
3	M4/5	M5/5	M0/5	M1/3	M2/8	M3/9
4	M5/9	M0/3	M1/5	M2/4	M3/3	M4/1
5	M0/3	M1/3	M2/9	M3/10	M4/4	M5/1

Fonte: Adaptado de Fisher e Thompson (1963)

Tabela A.2: *Instância LA01 — 10 jobs, 5 máquinas (Melhor conhecido: 666)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/1	M1/3	M2/6	M3/7	M4/3
1	M0/8	M1/5	M2/10	M3/10	M4/10
2	M0/5	M1/4	M2/8	M3/9	M4/1
3	M0/5	M1/5	M2/5	M3/3	M4/8
4	M0/9	M1/3	M2/5	M3/4	M4/3
5	M0/3	M1/3	M2/9	M3/10	M4/4
6	M0/3	M1/4	M2/8	M3/6	M4/2
7	M0/4	M1/3	M2/7	M3/9	M4/3
8	M0/6	M1/5	M2/7	M3/8	M4/4
9	M0/5	M1/3	M2/9	M3/2	M4/6

Fonte: Adaptado de Lawrence (1984)

Tabela A.3: *Instância LA02 — 10 jobs, 5 máquinas (Melhor conhecido: 655)*

job	Op1	Op2	Op3	Op4	Op5
0	M1/21	M2/53	M3/14	M0/65	M4/59
1	M1/83	M2/34	M0/33	M4/52	M3/46
2	M2/77	M3/61	M4/12	M0/33	M1/61
3	M0/58	M2/96	M3/76	M4/11	M1/44
4	M1/52	M2/51	M4/61	M3/46	M0/49
5	M2/63	M0/35	M1/17	M4/80	M3/51
6	M2/52	M4/31	M1/48	M0/67	M3/44
7	M4/26	M0/69	M2/55	M3/27	M1/20
8	M1/60	M4/49	M0/66	M2/36	M3/47
9	M3/20	M0/45	M1/28	M2/33	M4/19

Fonte: Adaptado de Lawrence (1984)

Tabela A.4: *Instância LA03 — 10 jobs, 5 máquinas (Melhor conhecido: 597)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/25	M1/10	M2/35	M3/10	M4/20
1	M0/80	M1/20	M2/10	M3/55	M4/55
2	M0/10	M1/50	M2/15	M3/40	M4/30
3	M0/55	M1/60	M2/15	M3/20	M4/20
4	M0/10	M1/50	M2/35	M3/60	M4/30
5	M0/50	M1/35	M2/30	M3/40	M4/10
6	M0/15	M1/25	M2/30	M3/10	M4/15
7	M0/65	M1/45	M2/10	M3/55	M4/20
8	M0/30	M1/30	M2/20	M3/20	M4/45
9	M0/60	M1/15	M2/25	M3/45	M4/15

Fonte: Adaptado de Lawrence (1984)

Tabela A.5: *Instância LA04 — 10 jobs, 5 máquinas (Melhor conhecido: 590)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/18	M1/2	M2/6	M3/18	M4/16
1	M0/6	M1/16	M2/12	M3/2	M4/16
2	M0/14	M1/6	M2/12	M3/2	M4/14
3	M0/18	M1/6	M2/4	M3/2	M4/12
4	M0/10	M1/10	M2/12	M3/10	M4/16
5	M0/14	M1/8	M2/6	M3/16	M4/18
6	M0/14	M1/6	M2/6	M3/4	M4/16
7	M0/10	M1/18	M2/2	M3/16	M4/4
8	M0/10	M1/8	M2/12	M3/12	M4/18
9	M0/6	M1/4	M2/10	M3/10	M4/12

Fonte: Adaptado de Lawrence (1984)

Tabela A.6: *Instância LA05 — 10 jobs, 5 máquinas (Melhor conhecido: 545)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/6	M1/10	M2/6	M3/10	M4/8
1	M0/8	M1/10	M2/10	M3/2	M4/4
2	M0/10	M1/10	M2/8	M3/2	M4/6
3	M0/10	M1/8	M2/4	M3/8	M4/4
4	M0/10	M1/4	M2/6	M3/8	M4/10
5	M0/10	M1/8	M2/6	M3/2	M4/10
6	M0/8	M1/8	M2/2	M3/6	M4/10
7	M0/6	M1/2	M2/6	M3/10	M4/8
8	M0/10	M1/8	M2/2	M3/6	M4/10
9	M0/6	M1/2	M2/2	M3/8	M4/10

Fonte: Adaptado de Lawrence (1984)

Tabela A.7: *Instância LA06 — 10 jobs, 5 máquinas (Melhor conhecido:926)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/11	M1/10	M2/9	M3/7	M4/8
1	M0/6	M1/5	M2/12	M3/6	M4/10
2	M0/16	M1/11	M2/15	M3/10	M4/9
3	M0/10	M1/6	M2/11	M3/10	M4/12
4	M0/9	M1/12	M2/8	M3/7	M4/10
5	M0/10	M1/9	M2/13	M3/13	M4/9
6	M0/11	M1/10	M2/8	M3/10	M4/13
7	M0/9	M1/7	M2/9	M3/12	M4/10
8	M0/8	M1/10	M2/10	M3/9	M4/9
9	M0/13	M1/12	M2/12	M3/9	M4/13

Fonte: Adaptado de Lawrence (1984)

Tabela A.8: *Instância LA07 — 15 jobs, 5 máquinas (Melhor conhecido:890)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/11	M1/10	M2/9	M3/7	M4/8
1	M0/6	M1/5	M2/12	M3/6	M4/10
2	M0/16	M1/11	M2/15	M3/10	M4/9
3	M0/10	M1/6	M2/11	M3/10	M4/12
4	M0/9	M1/12	M2/8	M3/7	M4/10
5	M0/10	M1/9	M2/13	M3/13	M4/9
6	M0/11	M1/10	M2/8	M3/10	M4/13
7	M0/9	M1/7	M2/9	M3/12	M4/10
8	M0/8	M1/10	M2/10	M3/9	M4/9
9	M0/13	M1/12	M2/12	M3/9	M4/13
10	M0/10	M1/11	M2/10	M3/10	M4/12
11	M0/9	M1/9	M2/14	M3/7	M4/10
12	M0/10	M1/11	M2/11	M3/10	M4/12
13	M0/10	M1/10	M2/8	M3/10	M4/10
14	M0/13	M1/11	M2/9	M3/11	M4/11

Fonte: Adaptado de Lawrence (1984)

Tabela A.9: *Instância LA08 — 15 jobs, 5 máquinas (Melhor conhecido: 863)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/11	M1/10	M2/9	M3/7	M4/8
1	M0/6	M1/5	M2/12	M3/6	M4/10
2	M0/16	M1/11	M2/15	M3/10	M4/9
3	M0/10	M1/6	M2/11	M3/10	M4/12
4	M0/9	M1/12	M2/8	M3/7	M4/10
5	M0/10	M1/9	M2/13	M3/13	M4/9
6	M0/11	M1/10	M2/8	M3/10	M4/13
7	M0/9	M1/7	M2/9	M3/12	M4/10
8	M0/8	M1/10	M2/10	M3/9	M4/9
9	M0/13	M1/12	M2/12	M3/9	M4/13
10	M0/10	M1/11	M2/10	M3/10	M4/12
11	M0/9	M1/9	M2/14	M3/7	M4/10
12	M0/10	M1/11	M2/11	M3/10	M4/12
13	M0/10	M1/10	M2/8	M3/10	M4/10
14	M0/13	M1/11	M2/9	M3/11	M4/11

Fonte: Adaptado de Lawrence (1984)

Tabela A.10: *Instância LA09 — 15 jobs, 5 máquinas (Melhor conhecido: 951)*

job	Op1	Op2	Op3	Op4	Op5
0	M0/11	M1/10	M2/9	M3/7	M4/8
1	M0/6	M1/5	M2/12	M3/6	M4/10
2	M0/16	M1/11	M2/15	M3/10	M4/9
3	M0/10	M1/6	M2/11	M3/10	M4/12
4	M0/9	M1/12	M2/8	M3/7	M4/10
5	M0/10	M1/9	M2/13	M3/13	M4/9
6	M0/11	M1/10	M2/8	M3/10	M4/13
7	M0/9	M1/7	M2/9	M3/12	M4/10
8	M0/8	M1/10	M2/10	M3/9	M4/9
9	M0/13	M1/12	M2/12	M3/9	M4/13
10	M0/10	M1/11	M2/10	M3/10	M4/12
11	M0/9	M1/9	M2/14	M3/7	M4/10
12	M0/10	M1/11	M2/11	M3/10	M4/12
13	M0/10	M1/10	M2/8	M3/10	M4/10
14	M0/13	M1/11	M2/9	M3/11	M4/11

Fonte: Adaptado de Lawrence (1984)

Tabela A.11: *Instância LA10 — 10 jobs, 10 máquinas (Melhor conhecido: 940)*

Job	**Op1**	**Op2**	**Op3**	**Op4**	**Op5**	**Op6**	**Op7**	**Op8**	**Op9**	**Op10**
0	M0/29	M1/78	M2/9	M3/36	M4/49	M5/11	M6/89	M7/60	M8/84	M9/48
1	M0/43	M1/90	M2/75	M3/11	M4/69	M5/28	M6/46	M7/46	M8/72	M9/30
2	M0/91	M1/85	M2/39	M3/74	M4/90	M5/10	M6/12	M7/88	M8/77	M9/33
3	M0/81	M1/46	M2/84	M3/62	M4/88	M5/58	M6/71	M7/51	M8/27	M9/14
4	M0/14	M1/33	M2/21	M3/37	M4/51	M5/70	M6/25	M7/51	M8/41	M9/76
5	M0/32	M1/90	M2/29	M3/68	M4/49	M5/59	M6/58	M7/41	M8/17	M9/82
6	M0/48	M1/32	M2/57	M3/67	M4/22	M5/98	M6/37	M7/26	M8/10	M9/71
7	M0/55	M1/77	M2/59	M3/90	M4/76	M5/59	M6/47	M7/88	M8/17	M9/84
8	M0/77	M1/72	M2/77	M3/78	M4/49	M5/71	M6/42	M7/75	M8/81	M9/73
9	M0/38	M1/38	M2/69	M3/39	M4/34	M5/60	M6/25	M7/49	M8/49	M9/64

Adaptado de Lawrence (1984).

Fonte: Adaptado de Lawrence (1984)