Representações de Algoritmos Genéticos para o Problema de Escalonamento Estático de Tarefas em Multiprocessadores

Eduardo Cassiano da Silva

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2019

Eduardo Cassiano da Silva

Representações de Algoritmos Genéticos para o Problema de Escalonamento Estático de Tarefas em Multiprocessadores

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Paulo Henrique Ribeiro Gabriel

Uberlândia
2019

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

S586 Silva, Eduardo Cassiano da, 1992-
2020 Representações de Algoritmos Genéticos para o Problema de Escalonamento Estático de Tarefas em Multiprocessadores [recurso eletrônico] / Eduardo Cassiano da Silva. - 2020.

Orientador: Paulo Henrique Ribeiro Gabriel.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2020.104
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Gabriel, Paulo Henrique Ribeiro,1984-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certi cam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Representações de Algoritmos Genéticos para o Problema de Escalonamento Estático de Tarefas em Multiprocessadores**" por **Eduardo Cassiano da Silva** como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Uberlândia, 20 de Janeiro de 2020

Orientador: _____
Prof. Dr. Paulo Henrique Ribeiro Gabriel
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Murillo Guimarães Carneiro
Universidade Federal de Uberlândia

_____
Profª. Drª. Telma Woerle de Lima Soares
Universidade Federal de Goiás

Dedico este estudo à minha família por ter
me presenteado com sua paciência, compreensão e amor.

Agradecimentos

Essa dissertação de mestrado é fruto da contribuição do precioso apoio de diversas pessoas.

Primeiramente, direciono meus agradecimento ao professor Paulo Henrique, por toda a paciência, confiança e dedicação na realização deste trabalho. Agradeço muitíssimo pelas correções, pela motivação, pela oportunidade e por todo o amparo cedido.

Venho também agradecer a todos os meus colegas que me acompanharam nessa jornada desde o começo. À Danielli Lima, sou imensamente grato pelos conselhos, alertas e por continuar me motivando a trilhar esse campo. Da mesma forma, agradeço ao Thiago Fialho pelo companheirismo no decorrer das disciplinas e por ter me ajudado com diversas ideias e sugestões. Não poderia deixar de agradecer à Maria Eugênia pelo seu apoio e por sua sensatez vital nos momentos mais difíceis.

Também agradeço à Fundação de Amparo à Pesquisa de Minas Gerais (FAPEMIG) pelo fomento e suporte na execução desse projeto. Semelhantemente, sou grato aos professores da Faculdade de Computação (FACOM) e por todo o suporte oferecido pela Secretaria do Programa de Pós-Graduação em Computação. Estendo meus agradecimentos ao professor Murillo Guimarães e à professora Telma Woerle pela disponibilidade e cooperação para a realização da banca examinadora.

Por fim, sou eternamente grato à minha família pelo apoio incondicional, pela compreensão e paciência nestes últimos anos. A todos que contribuíram para a construção e finalização desse projeto, o meu muito obrigado.

"A future is not given to you. It is something you must take for yourself."
(Pod 042 para Pod 153)

Resumo

Este trabalho apresenta uma revisão sistemática da literatura sobre as características dos algoritmos genéticos aplicado ao problema do escalonamento de tarefas em sistemas multiprocessados. A revisão foi aplicada sobre trabalhos publicados entre 1990 e 2018. Sendo assim, um protocolo de pesquisa foi elaborado para definir as fases de execução da revisão, assim como as regras para incluir e excluir estudos. Após a execução da revisão, 13 representações de cromossomos foram identificadas. Dadas as particularidades de cada representação, 78 operadores genéticos foram encontrados. As funções objetivo empregadas nesses algoritmos também foram sumarizadas. Por fim, as quatro codificações (Codificação de Listas Topológicas (TLE), Codificação de Alocação e Escalonamento (MSE), Codificação por Lista Ordenada (OLE) e Codificação por Lista de Processadores (PLE)) foram comparadas através de uma análise de dispersão. Desse modo, ao aplicar as métricas de makespan, flowtime e load balance, foi possível observar um melhor desempenho na representação OLE e uma equivalência entre TLE, MSE e PLE. Por fim, a síntese elaborada nesta dissertação pode ser significativa na tomada de decisão e na realização de novas pesquisas.

Palavras-chave: Algoritmos Genéticos; Escalonamento de Tarefas, Sistemas Multiprocessados; Representações; Função Objetivo; DAG.

Abstract

This dissertation presents a systematic review of the literature on the characteristics of genetic algorithms applied to the multiprocessor task scheduling problem. The review was applied on works published between 1990 and 2018. A research protocol was developed to define the stages of the review as well as the rules for including and excluding studies. In the review, 13 representations of chromosomes were identified. Given the particularities of each representation, 78 genetic operators were found. The objective functions employed in these algorithms have also been summarized. Finally, the four encodings (Topological List Encoding (TLE), Matching Scheduling Encoding (MSE), Ordered List Encoding (OLE) and Processor List Encoding (PLE)) were compared by dispersion analysis. Thus, when applying the metrics of makespan, flowtime and load balance, it was possible to observe a better performance in the OLE representation and an equivalence between TLE, MSE and PLE. Lastly, the synthesis developed in this dissertation can be significant in decision making and in conducting new research.

Keywords: Genetic Algorithms; Task Scheduling, Multiprocessed Systems; Representations; Objective function; DAG.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ASPX *Activity Single Point Crossover*

APM *Activity Processor Mutation*

CX *Cycle Crossover*

CM *Complementation Mutation*

DDP *Data Dominating Parent*

DM *Duplication Mutation*

DRM *Digraph Reorder Mutation*

DAT *Data Available time*

DLE *Double Level Encoding*

DVFSE *Dynamic Voltage Frequency Scaling Encoding*

DVFSM *Dynamic Voltage Frequency Scaling Mutation*

FX *Fusion Crossover*

FAM *First Available Machine*

FTE *Fixed Task Encoding*

GA *Genetic Algorithm*

GX *Grouping Crossover*

HSM *Height Swap Mutation*

HCM *Height Change Mutation*

IHM *Internal Height Mutation*

ISM *Idle Swap Mutation*

IM *Invert Mutation*

IX *Inner Crossover*

LM *List Mutation*

MISF *Most Immediate Successors First*

MX *Mask Crossover*

ME *Matrix Encoding*

MLE *Mapping List Encoding*

MM *Mapping Mutation*

MSE *Matching Scheduling Encoding*

MSSPX *Main Sequence Single Point Crossover*

MSSM *Main Sequence Swap Mutation*

MPX *Multipoint Crossover*

OX *Order Crossover*

OLE *Ordered List Encoding*

PMX *Partially Matched Crossover*

PSM *Processor Swap Mutation*

PM *Processor Mutation*

PLE *Processor List Encoding*

PAM *Priority Allocation Mutation*

PSPX *Partition Single Point Crossover*

PX *Position Crossover*

PE *Priority Encoding*

PTM *Priority Task Mutation*

QoS *Quality of Service*

QE *Q-bit Encoding*

QS *Questão Secundária*

QP *Questão Principal*

RM *Removal Mutation*

RPX *Random Point Crossover*

RPM *Random Probability Mutation*

RCM *Recovery Mutation*

RE *Range Encoding*

SM *Swap Mutation*

SRM *Sort Mutation*

SPX *Single Point Crossover*

STM *Shift Task Mutation*

SFTM *Shuffle Task Mutation*

SPM *Shuffle Priority Mutation*

SBX *Simulated Binary Crossover*

TLE *Topological List Encoding*

TD *Task Duplication*

TEOL *Task Executation Order List*

TTX *Target Task Crossover*

TEOLX *Task Execution Order List Crossover*

TPX *Two Point Crossover*

TE *Tuple Encoding*

TM *Task Mutation*

UX *Uniform Crossover*

UDM *Uniform Distributation Mutation*

WMX *Weight Mapping Crossover*

Lista de símbolos

Δ Diferença entre as variáveis de decisão de ambos os pai.

γ Velocidade de instrução.

λ Quantidade de conexões.

ϕ Quantidade possível de mapeamentos.

π Decodificação de cada gene i.

ac_i Valor de atividade de uma tarefa i.

D^l Conjunto de parede DVFS de um processador p_l

g_i Gene i de um cromossomo.

K Conjunto de todos os pares DVFS

$p_{c_{dec}}$ Probabilidade de uma variável de decisão participar de uma recombinação

$p_{c_{mem}}$ Probabilidade de um par de cromossomos participar de uma recombinação

pai_val Valor da variável de decisão de um cromossomo pai.

r Tipos de máquina.

R_s Confiabilidade de um escalonamento s.

rdr Taxa de falha.

s_k Velocidade relativa do processador

$v\_inferior$ Valor mínimo alcançado por uma variável de decisão.

$v\_superior$ Valor máximo alcançado por uma variável de decisão.

v_k Voltagem operacional do processador

Lista de algoritmos

Sumário

Capítulo 1

Introdução

A atividade de designar tarefas a recursos que as executem é aplicada em variadas áreas de planejamento. Existem muitas versões de problemas baseados nessa premissa. Por exemplo, ao dividirem um sistema de processamento, os diversos processos relacionados devem obedecer a uma ordem de execução pré-estabelecida, de forma a proporcionar a sua completa execução. Essa sequência de execução tem sua complexidade aumentada quando os processos encontram um cenário com múltiplos processadores, seja por unidades processadoras multinúcleos ou por arranjos de computadores interligados em rede. A atividade de organizar, designar e avaliar a sequência de execução das tarefas sob um conjunto de recursos é conhecida como escalonamento. Nesses ambientes, seja para otimizar uma métrica de desempenho, tal como a utilização do sistema ou o tempo de finalização, uma aplicação paralela deve ser alocada sobre um conjunto de recursos computacionais disponíveis (processadores) (HOU; ANSARI; REN, 1994; SHEIKH; AHMAD; ARSHAD, 2017).

O escalonamento de tarefas em multiprocessadores (MTSP[1]) é um problema da classe NP-difícil (GOLUB; KASAPOVIC, 2002). Além disso, esse tipo de problema pode apresentar uma extensa variação de cenários baseados em restrições atribuídas às tarefas ou aos recursos. Arranjos de computadores homogêneos (isto é, sem variações de hardware) ou heterogêneos são diferentes exemplos de cenários. Outro exemplo é o custo de comunicação, relacionado às tarefas, que contabiliza o esforço necessário entre tarefas que devem se comunicar no decorrer da execução. Além das restrições, o problema também pode apresentar métricas que medem o desempenho do escalonador em satisfazer os objetivos. É natural que as métricas empregadas possuam objetivos conflitantes e consequentemente, aumentem a complexidade, gerando assim novas classes de problema baseados na premissa do escalonamento.

Por ser um problema constantemente abordado na literatura, existem diversas soluções implementadas por meio de heurísticas e meta-heurísticas. Sobre essa última proposta, os

[1] Do inglês, *Multiprocessor Task Scheduling*.

Algoritmos Genéticos (GAs[2]) são ferramentas comumente utilizadas (SHEIKH; AHMAD; ARSHAD, 2017; AKBARI, 2018; PILLAI et al., 2018). GAs representam uma classe de técnicas bioinspiradas que podem ser adaptados em vários tipos de problemas. Ao se inspirar no paradigma evolucionista, GAs trabalham com a evolução de várias soluções simultaneamente, garantindo uma vantagem significativa comparada a técnicas sequenciais. Dessa forma, GAs podem explorar a computação paralela e, portanto, apresentarem melhor desempenho. Os operadores genéticos são outras propriedades importantes nos GAs uma vez que combinam ou modificam soluções, expandindo a capacidade de exploração do espaço de soluções.

Com uma quantidade de trabalhos significativos sobre o tema, existem variadas taxonomias voltadas a enumerar as heurísticas e meta-heurísticas empregadas ao problema, bem como a evolução dessas técnicas no decorrer dos anos. Esses trabalhos são relevantes, uma vez que podem oferecer um amplo sumário de soluções utilizadas. Todavia, tais estudos podem não apresentar características inerentes às técnicas levantadas. Essas propriedades podem ser relevantes ao serem observadas em conjunto, de tal forma que seja possível a elaboração de novas propostas baseadas em alterações ou combinações dessas características. Uma maneira de investigar, sumarizar e sintetizar esses dados é através de uma Revisão Sistemática da Literatura (SRL[3]) (KITCHENHAM et al., 2009). A SLR consiste em um procedimento rigoroso de seleção e análise de trabalhos, de tal forma que seja possível responder às questões de pesquisa formuladas sem seguir um viés por parte dos pesquisadores, ou seja, sem apresentar alguma tendência.

Esta dissertação de mestrado propõe a aplicação de técnicas de SLR para extração das codificações, funções objetivos e métricas utilizadas em problemas de escalonamento em sistemas multiprocessados. Um levantamento dessas propriedades permite uma completa avaliação através de comparações de resultados ou medição de custos de desempenho. Com a avaliação desses itens, é possível elaborar futuras combinações de técnicas no intuito de criar melhores modelos evolutivos para a solução do problema. Por fim, também torna-se possível mapear quais técnicas implicam significantemente em melhores resultados e observar se a literatura segue algum viés de aplicação.

Com a conclusão deste trabalho, uma coleção de 63 estudos foi formada com a execução das técnicas de SLR. Em seguida, 13 representações empregadas pelos GAs na resolução do MTSP foram extraídas, contando também com 39 mecanismos de recombinação e outras 39 técnicas de mutação. Desse modo, utilizando os dados extraídos, essa dissertação elaborou uma ampla síntese de resoluções do MTSP. Além disso, quatro representações (Codificação de Alocação e Escalonamento, Codificação de Lista Topológica, Codificação de Lista de Processadores e Codificação de Lista Ordenada) foram selecionadas e analisadas considerando três métricas de comparação (makespan, flowtime, load

[2] Do inglês, *Genetic Algorithm*.
[3] Do inglês, *Systematic Literature Review*.

balance). Ao final, considerando um experimento de geração de população inicial e análise de dispersão, foi possível observar um comportamento equivalente entre a Codificação de Lista Topológica, Codificação de Lista de Processadores e a Codificação de Alocação e Escalonamento. Também foi possível observar um desempenho melhor na Codificação de Lista Ordenada em todas as métricas aplicadas, apesar desta utilizar uma heurística de alocação de tarefas, o que pode reduzir a diversidade populacional e, consequentemente, dificultar o processo de busca do GA.

1.1 Hipótese

Propriedades dos GAs como codificações dos cromossomos, operadores genéticos e funções objetivos podem influenciar diretamente na resolução de problemas. Dessa forma, no MTSP, a escolha da codificação de cromossomo influencia diretamente no desempenho da solução, mesmo sem a empregabilidade dos operadores genéticos.

1.2 Justificativa

Geralmente, o principal objetivo das taxonomias é sumarizar todos os trabalhos referentes a um tema. Dessa forma, um levantamento pode deixar de abordar características específicas das técnicas. Portanto, no caso do MTSP, um levantamento combinado a um estudo pode destacar propriedades intrínsecas dos GAs bem como as métricas abordadas no problema. Os GAs contemplam um conjunto variado de técnicas a fim de facilitarem a exploração do espaço de soluções. As características de um GA podem prover variadas modificações que criam novas possibilidades com melhores chances de alcançarem bons resultados. Portanto, este estudo visa levantar e organizar em uma síntese as características inerentes ao operadores genéticos, representações dos indivíduos, funções objetivo e as métricas empregadas ao problema. Este estudo também, considerando variadas categorias de codificações, propõe-se a analisar e comparar as principais representações de indivíduos empregadas ao problema em diferentes métricas de desempenho.

1.3 Objetivos

1.3.1 Objetivo Geral

Este trabalho tem como objetivo principal aplicar uma Revisão Sistemática da Literatura para sumarizar e, posteriormente, medir aspectos específicos relacionados ao problema de escalonamento de tarefas em multiprocessadores, onde as técnicas de resolução utilizadas são os Algoritmos Genéticos. Dessa forma, ao analisar trabalhos da literatura disponíveis entre a 1990 e 2018, o propósito é reunir e sumarizar as representações de

indivíduos, as funções objetivo, as métricas associadas ao problema e os operadores genéticos utilizados. A partir desse estudo, busca-se analisar possíveis relações entre algumas codificações e métricas empregadas ao problema. Dessa forma, a síntese e os resultados gerados podem orientar novas investigações em técnicas promissoras menos exploradas ou possíveis novas combinações.

1.3.2 Objetivos Específicos

Os objetivos específicos propostos por este trabalho são:

- ❏ Elaborar uma taxonomia através de um levantamento da literatura, entre os anos de 1990 e 2018, utilizando os rigorosos procedimentos da SRL e que contemple as representações, funções objetivo, operadores genéticos e métricas utilizadas na resolução do MTSP com GAs;
- ❏ Analisar os trabalhos levantados pela SRL, de forma a investigar possíveis relações inerentes a essas publicações;
- ❏ Construir uma biblioteca que contemple codificações dos itens sumarizados pela SRL, de maneira a atuar como um inventário que facilite a execução de experimentos voltados a unificação de técnicas ou comparações entre as mesmas;

1.4 Organização da Dissertação

Essa dissertação foi estruturada sobre as seguintes divisões:

Capítulo 2 – Fundamentação Teórica: Sintetiza uma revisão dos principais temas relacionados a essa pesquisa, sendo esses o Problema do Escalonamento de Tarefas em Multiprocessadores, os Algoritmos Genéticos e a Revisão Sistemática da Literatura;

Capítulo 3 – Revisão Sistemática sobre GAs aplicados ao MTSP: Inclui o protocolo elaborado e utilizado na execução da Revisão Sistemática da Literatura. Também são apresentadas as diversas informações levantadas correspondentes às questões de pesquisa definidas no protocolo;

Capítulo 4 – Representações: Sumariza todas as representações de cromossomos empregadas nos trabalhos levantados na revisão. Os operadores genéticos, recombinação e mutação, também são sumarizados em cada representação apresentada;

Capítulo 5 – Funções Objetivo: Agrupa todas as funções objetivos levantadas nos trabalhos encontrados na revisão. As funções são apresentadas seguindo uma hierarquia baseada nas codificações de cromossomo pesquisadas.

Capítulo 6 – Experimentos: Contempla uma coleção de experimentos realizados sobre as principais codificações levantadas. Características do experimento, análises de dispersão e a metologia empregada são apresentadas.

Capítulo 7 – Conclusão: Apresenta observações e as principais conclusões desta dissertação. Ao final, são sugeridos alguns possíveis trabalhos futuros a curto e médio prazo.

Capítulo 2

Fundamentação Teórica

Este capítulo contempla o material necessário e adequado à compreensão das técnicas utilizadas e estudadas neste trabalho. Desta forma, diversos conceitos essenciais são resumidos e apresentados.

A Seção 2.1 introduz o problema do escalonamento estático de tarefas em multiprocessadores. A Seção 2.2 traz uma descrição geral sobre os conceitos de algoritmos genéticos. Finalmente, a Seção 2.3 apresenta conceitos relacionados à técnica de revisão sistemática da literatura, juntamente com exemplos de aplicações e possíveis divisões necessárias à execução da revisão.

2.1 Escalonamento Estático de Tarefas em Multiprocessadores

O Escalonamento de Tarefas em Multiprocessadores (MTSP) consiste na distribuição e execução de tarefas relacionadas a um conjunto interligado de processadores. Essas tarefas podem ser formadas, por exemplo, através da divisão de uma tarefa grande em pequenas tarefas correlacionadas (SINGH; SINGH, 2012). Existem variações do problema, baseando-se nas características das tarefas e na quantidade de informação disponível (KAUR; CHHABRA; SINGH, 2010a). Outras variações podem estar presentes no tratamento do envio de dados entre tarefas relacionadas e nos aspectos do sistema de processamento adotado. Neste contexto, o MTSP estático é estruturado sobre um modelo computacional onde a relação de dependências e a ordem de execução entre as tarefas são conhecidas a priori e não mudam no decorrer do escalonamento ou na execução das tarefas (XU et al., 2014). Além disso, enquanto algumas tarefas dependem da saída de outras, outras podem executar independentemente ao mesmo tempo, o que aumenta o paralelismo no problema (GUPTA; KUMAR; AGARWAL, 2010).

O MTSP é um problema amplamente explorado na literatura, seja por heurísticas ou meta-heurísticas, além de ser um problema NP-difícil. Golub e Kasapovic (2002) sin-

tetizam na Tabela 1 a elevação de complexidade do MTSP em relação a quantidade de processadores, tempo de processamento e restrições de precedência. A tabela revela que, apesar das tarefas possuírem custo de execução igual, o problema atinge uma complexidade NP-difícil quando as restrições e a quantidade de processadores tornam-se arbitrárias. Além disso, essa complexidade é elevada quando o custo de execução também torna-se arbitrário.

Tabela 1 – Complexidade de problemas de escalonamento.

Processadores (m)	**Tempo p/ Tarefa**	**Restrições**	**Complexidade**
Arbitrária	Igual	Árvore	O(n)
2	Igual	Arbitrária	O(n^2)
Arbitrário	Igual	Arbitrária	NP-difícil
Fixado($m \geq 2$)	$t_i = 1$ ou 2 $\forall i$	Arbitrária	NP-difícil
Arbitrária	Arbitrária	Arbitrária	Fortemente NP-difícil

Fonte: adaptado de Golub e Kasapovic (2002).

As próximas subseções apresentam mais detalhadamente aspectos do MTSP estático aplicado em ambientes homogêneos, heterogêneos e com controle dinâmico de frequência.

2.1.1 Representação das Tarefas

A relação de precedência entre as tarefas pode ser modelada conforme um grafo acíclico dirigido (DAG[1]), $G = (T, E)$, onde $T = t_1, t_2, ...t_n$ é o conjunto de vértices que representa cada tarefa e E é o conjunto de arestas que indica a ordem de precedência. Desse modo, dado as tarefas t_i e $t_j \in T$, se $e_{ij} \in E$, então t_j só poderá começar sua execução após a conclusão de t_i (OMARA; ARAFA, 2009). Com a representação em grafo, as expressões $succ(t_i)$ e $pred(t_i)$ são geralmente utilizadas para representar, respectivamente, os sucessores imediatos e os antecessores imediatos de uma tarefa t_i.

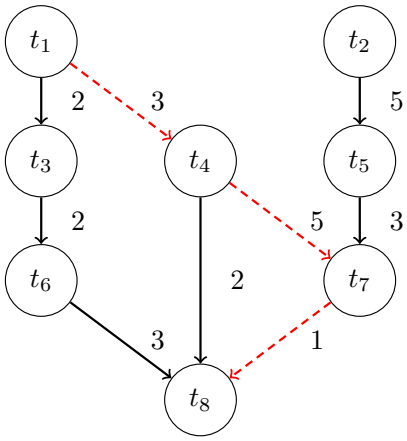
No DAG, o peso de cada vértice representa a quantidade de ciclos de processamento requeridos por uma tarefa para ser executada, $et(ti) > 0$, e o peso em cada aresta demonstra o custo computacional de comunicação em transferir dados de uma tarefa t_i para sua sucessora t_j (DHINGRA; GUPTA; BISWAS, 2014). Definido como $c(t_i, t_j) > 0$, o custo de comunicação não é computado quando as tarefas antecessora e sucessora estão escalonadas no mesmo processador (TSUCHIYA; OSADA; KIKUNO, 1998). Outra forma de reduzir ou eliminar o custo de comunicação é a duplicação de tarefas, em que algumas cópias de tarefas são distribuídas aos demais processadores do conjunto, o que elimina a necessidade de transferências externas (TSUCHIYA; OSADA; KIKUNO, 1997).

A Figura 1 apresenta uma instância do MTSP. Primeiramente, na Figura 1(a), um DAG com 8 tarefas ($n = 8$) é ilustrado. As arestas em destaque identificam o caminho

[1] Do inglês, *Directed Acyclic Graph.*

crítico, sendo esse o caminho com a soma máxima de custo de execução e comunicação entre as tarefas de entrada e saída (OMARA; ARAFA, 2009). Outro indicador semelhante ao caminho crítico é a Sequência Principal (MS[2]). Conforme a definição de Yao, You e Li (2004), uma $MS(t_i)$ indica qual é a maior soma de custo de computação de uma tarefa de entrada até uma tarefa t_i. Dentre todas as MSs, a sequência com o maior custo computacional é chamada de sequência principal do DAG (DAGMS). Em seguida, a Figura 1(b) relaciona o custo de computação ($et(t_i)$), a quantidade total de dados transferidos a cada tarefa ($TCC(t_i)$) e o valor de altura no grafo.

Figura 1 – Uma instância do MTSP com 8 tarefas.



(a) DAG com 8 tarefas.

Tarefas	$et(t_i)$	$TCC(t_i)$	$altura(i)$
t_1	3	0	1
t_2	2	0	1
t_3	1	2	2
t_4	2	3	2
t_5	3	4	2
t_6	2	2	3
t_7	2	8	3
t_8	4	6	4

(b) Propriedades das tarefas.

Fonte: adaptado de Hou, Ansari e Ren (1994) e de Tsuchiya, Osada e Kikuno (1997).

Conforme apresentado na Figura 1, as tarefas dispostas no DAG podem apresentar um valor relacionado à altura. Obtido conforme a Expressão 1, em que $pred(t_i)$ representa as tarefas predecessoras de uma tarefa t_i, o valor de altura é utilizado como um dos possíveis parâmetros na definição de uma ordem válida de execução (HOU; ANSARI; REN, 1994). Naturalmente, as tarefas com um valor de altura menor tendem a ser priorizadas na organização da execução.

$$altura(t_i) = \begin{cases} 0 & \text{, se } pred(t_i) = \emptyset \\ 1 + \max\limits_{t_j \in pred(t_i)} \{altura(t_j)\} & \text{, caso contrário} \end{cases} \quad (1)$$

Além da função altura, existem outros índices utilizados na ordenação das tarefas. Hou, Ansari e Ren (1994) propõem uma modificação na função altura para flexibilizar o índice de tarefas distribuídas em intervalos longos no DAG. Outros índices populares são o *t-level* e o *b-level*. O primeiro é definido como o comprimento do mais longo caminho no DAG de uma tarefa de entrada até t_i, excluindo o custo de execução de t_i (KAUR;

[2] Do inglês, *Main Sequence*

CHHABRA; SINGH, 2010a). Analogamente, o índice *b-level* é definido como o mais longo caminho de t_i até uma tarefa de saída, incluindo o custo de execução de t_i (KAUR; CHHABRA; SINGH, 2010a). Os índices *t-level* e *b-level* são obtidos segundo as Expressões 2 e 3, respectivamente. Na Expressão 3, $succ(t_i)$ representa as tarefas que sucedem uma tarefa t_i.

$$tlevel(t_i) = \max_{t_j \in pred(t_i)} tlevel(t_j) + wj + c_{ij} \tag{2}$$

$$blevel(t_i) = wi + \max_{t_j \in succ(t_i)} c_{ij} + blevel(t_j) \tag{3}$$

2.1.2 Arquitetura do Ambiente

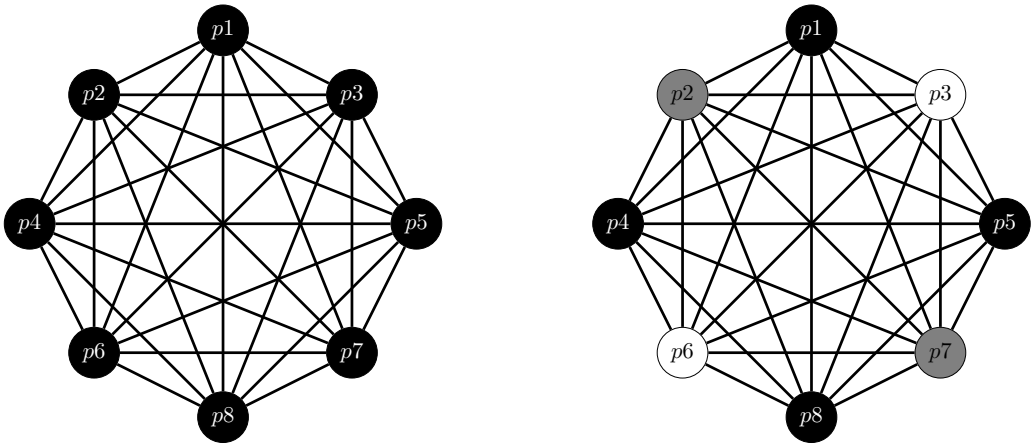
2.1.2.1 Cenários Clássicos

No MTSP, as tarefas dispostas no DAG devem ser distribuídas em um ambiente de computação com multiprocessadores, sendo este composto por um conjunto $P = p_1, p_2, ..., p_m$ de processadores. Os processadores dispostos nesse conjunto podem representar dois modelos de arquitetura: homogêneo e heterogêneo. Em um cenário com processadores homogêneos, o custo computacional de cada tarefa é o mesmo para qualquer um dos nós do sistema. Em contrapartida, heterogeneidade do sistema significa que os processadores possuem diferentes velocidades ou capacidades de processamento (SINGH; SINGH, 2012). O custo computacional de cada tarefa, considerando a quantidade de instruções necessárias para a execução, pode variar de processador a processador. Sendo assim, em uma abordagem estática do problema, uma matriz de custo de execução, de ordem $n \times m$, relaciona o custo computacional de cada tarefa a cada processador do sistema (AHMAD; MUNIR; NISAR, 2012).

Independente da arquitetura empregada, os processadores do sistema são interconectados. Essas conexões garantem a transferência de dados entre tarefas não escalonadas no mesmo processador. Os dados transferidos entre as conexões são relacionados na matriz de custo de comunicação. De ordem $n \times n$, essa matriz dispõe a quantidade de dados necessários para serem transmitidos entre uma tarefa t_i para uma tarefa t_j (KANG; ZHANG; CHEN, 2011).

A Figura 2 apresenta dois modelos de sistema com multiprocessadores totalmente conectados. A heterogeneidade é representada na Figura 2(b) através das cores dos nós, de forma que, dentre os oito nós do conjunto, existem três modelos diferentes de processador. Adicionalmente, a Tabela 3(a) apresenta a matriz de custo de execução do DAG da Figura 1(a) relacionada ao modelo heterogêneo da Figura 2 (b). De forma análoga, a Tabela 3(b) sintetiza uma matriz dos custos de comunicação do DAG da Figura 1(a) relacionada ao modelo homogêneo da Figura 2(a).

Figura 2 – Grafo de processadores com oito unidades amplamente interligadas.



(a) Processadores Homogêneos (b) Processadores Heterogêneos

Fonte: adaptado de Singh e Singh (2012).

Figura 3 – Matrizes relacionando execução e comunicação no MTSP estático.

Tarefas	p_{i1}	p_{i2}	p_{i3}
t_1	4	8	6
t_2	5	10	7
t_3	2	4	1
t_4	3	6	3
t_5	4	8	7
t_6	3	6	2
t_7	4	8	4
t_8	5	10	3

(a) Matriz de custo de execução.

Tarefas	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
t_1	-	2	-	3	-	-	-	-
t_2	-	-	-	-	5	-	-	-
t_3	-	-	-	-	-	2	-	-
t_4	-	-	-	-	-	-	5	2
t_5	-	-	-	-	-	-	3	-
t_6	-	-	-	-	-	-	-	3
t_7	-	-	-	-	-	-	-	1
t_8	-	-	-	-	-	-	-	-

(b) Matriz de custo de comunicação.

Fonte: do autor.

2.1.2.2 Escala de Frequência de Voltagem Dinâmica

No escalonamento consciente de energia, além de tentar reduzir o tempo de finalização, um algoritmo de escalonamento busca também minimizar o consumo energético. Portanto, um algoritmo pode considerar as disponíveis tecnologias de hardware para controle de energia. Neste cenário, Guzek et al. (2014) destaca duas tecnologias voltadas à economia de energia: a hibernação de recursos e a Escala de Frequência de Voltagem Dinâmica (DVFS[3]). Sheikh, Ahmad e Fan (2016) destacam o DVFS como uma das tecnologias de controle equipadas em chips multinúcleo, que podem ser exploradas por um escalonador de recursos para designar tarefas aos núcleos. Conjuntamente, Guzek et al. (2014) destaca que a mais comum e moderna tecnologia de circuitos, semicondutor de

[3] Do inglês, *Dynamic Voltage Frequency Scaling*.

metal-óxido complementar (CMOS[4]), possui uma função para controlar o suprimento de voltagem e frequência.

A importância do uso do DVFS deve-se à economia de energia, que pode ser obtida ao usar menores níveis de voltagem. Todavia, conforme Guzek et al. (2014), essa redução na frequência de operação pode aumentar o tempo necessário à execução, consequentemente impactando negativamente no QoS[5]. Em contrapartida, a técnica DVFS é mais adaptável a mudanças do que a hibernação de recursos, uma vez que o tempo de transição do DVFS é muito menor do que o apresentado pela hibernação de recursos, consequentemente afetando a capacidade de resposta e portanto o QoS.

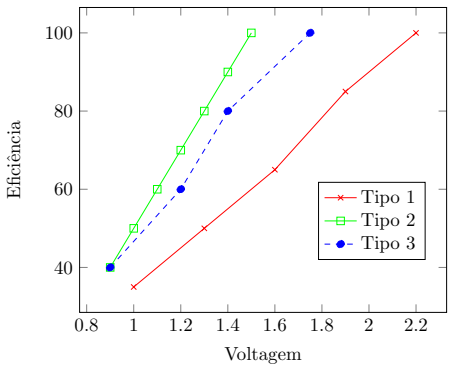
Segundo as definições de Guzek et al. (2014), cada processador p_l possui um conjunto de pares DVFS definido por D^l. Um par k de DVFS, denotado como d_k, é uma tupla (v_k, s_k), de forma que $v_k \in \mathbb{R}^+$ é a voltagem operacional do processador e $s_k \in \mathbb{R}^+$ é velocidade relativa formada pela relação da velocidade operacional para a velocidade máxima do processador.

A Figura 4(a) apresenta um exemplo, definido por Guzek et al. (2014) de uma relação de pares de DVFS em três processadores heterogêneos. Em contrapartida, o trabalho de Sheikh, Ahmad e Fan (2016) atribui a tecnologia DVFS aos níveis de frequência escalados em um grupo de processadores homogêneos. Em adição ao exemplo, o gráfico da Figura 4(b) apresenta a relação de linearidade entre voltagem e eficiência apontado por Guzek et al. (2014).

Figura 4 – Exemplo de pares DVFS em três processadores heterogêneos.

k	Tipo 1		Tipo 2		Tipo 3	
	V_k	RS_s	V_k	RS_s	V_k	RS_s
0	2.20	100	1.50	100	1.75	100
1	1.90	85	1.40	90	1.40	80
2	1.60	65	1.30	80	1.20	60
3	1.30	50	1.20	70	0.90	40
4	1.00	35	1.10	60		
5			1.00	50		
6			0.90	40		

(a) Relação Linear



(b) Processadores Heterogêneos

Fonte: adaptado de Guzek et al. (2014).

Apesar dos desafios encontrados no DVFS, tais como sobrecargas associadas e a inevitável energia estática, o mesmo continua predominante para controlar a energia de um chip. Em adição, o DVFS está sendo ativamente pesquisado enquanto é incorpo-

[4] Do inglês, *complementary metal-oxide-semiconductor*.
[5] Do inglês, *Quality of Service*.

rado em arquiteturas emergentes, possibilitando assim, controle por software ou hardware (SHEIKH; AHMAD; FAN, 2016).

2.1.3 Métricas Empregadas

O problema de MTSP pode apresentar vários critérios de otimização, sendo que os principais estão listados a seguir:

- **Makespan:** Com o objetivo de minimizá-lo, o tempo total de conclusão (*total completion time* ou *makespan*) pode ser obtido através da finalização da última tarefa executada ou da finalização do último processador em execução. O valor calculado é uma consequência direta dos tempos de execução das tarefas precedentes já executadas juntamente ao tempo de espera (ociosidade) encontrado nas comunicações ou lacunas no escalonamento. A Expressão 4 apresenta o cálculo do makespan, onde $ftp(p_j)$ determina o tempo de finalização de um processador t_j;

$$\text{makespan: } \{\max_{j \in N} ftp(p_j)\} \tag{4}$$

- **Flowtime:** Além do makespan, o tempo de finalização das tarefas pode estar associado a outra medida. Kaur, Chhabra e Singh (2010b) definem a minimização do flowtime como a tentativa de reduzir a soma dos tempos de finalização de todas as tarefas. A Expressão 5 define o cálculo do flowtime, sendo $ftt(t_j)$ o tempo de finalização de uma tarefa t_j.

$$\text{flowtime: } \{\sum_{j \in N} ftt(t_j)\} \tag{5}$$

- **Load Balance:** Outra forma de avaliar uma solução é observar se há uma distribuição adequada de tarefas nos processadores do ambiente. Essa distribuição pode ser avaliada pelo load balance (ou balançamento de carga). A Expressão 6 apresenta o cálculo do load balance, onde avg é a média dos tempos de finalização dos processadores do conjunto (OMARA; ARAFA, 2009):

$$\text{Load Balance : } \frac{\max_{j \in N} ftp(p_j)}{avg} \tag{6}$$

- **Confiabilidade:** Considerando um ambiente com um sistema sujeito a falhas, a métrica da confiabilidade mede a probabilidade de sucesso na execução de tarefas nesse sistema. O índice de confiabilidade pode ser definido conforme a Expres-

são 7 sendo que λ é uma constante, seguindo o modelo de distribuição de Poisson (CHITRA et al., 2010):

$$\text{Confiabilidade:} \sum_{j} \lambda_j ftp_j(s) \tag{7}$$

- **Consumo elétrico:** A Subseção 2.1.2.2 apresentou uma variação de cenário em que a velocidade de processamento dos processadores pode ser ajustada mediante um controle de frequência. Ao observar as frequências empregadas, é possível estimar a quantidade de energia gasta no sistema. Desse modo, a minimização do consumo elétrico é outra métrica possível;

- **Temperatura:** Conforme o consumo elétrico, a medição da temperatura do sistema é outra métrica que aborda sustentabilidade. De forma análoga, é possível estimar a temperatura dos núcleos de processamento ao observar as frequências adotadas em cada etapa do escalonamento;

- **Paralelismo:** Um escalonamento também pode ser medido pela capacidade de paralelismo. Sendo assim, é analisado o nível de distribuição das tarefas no sistema, de forma que, ao final, escalonamentos com um maior grau de paralelismo recebam melhores classificações;

- **Quantidade de processadores:** Um cenário que reduz a quantidade de recursos é uma outra possível forma de avaliar um escalonamento. Portanto, essa métrica mede soluções que tentam reproduzir um desempenho próximo ou equivalente a outras soluções em cenários com uma maior quantidade de processadores;

- **Custo de comunicação:** Além de ser proeminente em boa parte dos cenários abordados, o custo de comunicação também pode ser utilizado na otimização. Dessa forma, uma minimização do custo de comunicação significa a redução de espera de tarefas que necessitam de dados de suas tarefas predecessoras;

- **Utilização de recursos:** Um escalonamento também pode ser medido pela sua utilização dos recursos do sistema, isto é, pelo emprego equivalente dos processadores do conjunto. Em resumo, o desempenho na métrica é relacionado a ociosidade e a utilização dos processadores.

2.2 Algoritmos Genéticos

Algoritmos Genéticos (GAs) são técnicas de busca baseadas na simplificação de alguns processos evolutivos. A analogia deve-se ao mecanismo para geração de novos estados, em que uma seleção natural é representada através da reprodução sexuada entre duas ou mais

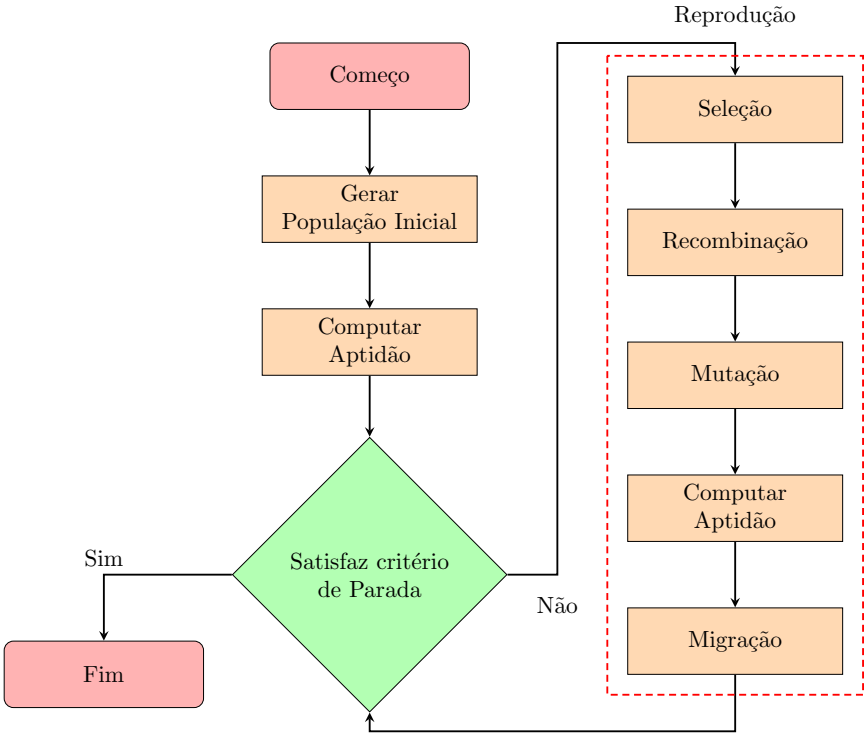
soluções. O surgimento da técnica está associada ao trabalho de Holland (1975), responsável por produzir a popularidade do método no início da década de 1970. GAs operam em uma população de soluções e não em uma única solução. Atuam implementando heurísticas tais como seleção (*selection*), recombinação (*crossover*) e mutação (*mutation*) para a evolução de melhores soluções (WALL, 1996). Em resumo, os GAs pertencem a uma classe de algoritmos probabilísticos que são implementados através de buscas paralelas e adaptativas baseadas na teoria evolutiva de sobrevivência dos mais aptos (PACHECO et al., 1999).

Os GAs pertencem a uma família de algoritmos voltada ao estado da solução e não ao custo do caminho para encontra-lá (RUSSELL; NORVIG; DAVIS, 2010). Em um problema de alocação de horários, o importante em sua resolução é encontrar uma configuração adequada que resolva a adversidade. Desse modo, a sequência de passos para a construção da solução não é o mais importante, mas sim, a solução final encontrada. Esta categoria de algoritmos é conhecida como "Busca Local", onde o objetivo é a aplicação de técnicas em problemas de otimização. Outros exemplos populares de técnicas nesta área são: Busca de subida de encosta (*Hill climbing search*), Têmpera simulada (*Simulated annealing*), Busca em feixe local (*Local Beam Search*) e Busca em feixe local estocástica (*Stochastic local beam search*).

A Figura 5 sintetiza o algoritmo de um modelo típico de GA. Ao início, o algoritmo cria uma lista de indivíduos, estes identificados como possíveis soluções do problema. Consequentemente, um cromossomo é a estrutura básica que codifica um indivíduo. Para diferenciar e classificar estes indivíduos, o algoritmo utiliza uma função de aptidão que atribui a cada indivíduo uma nota classificatória. Este valor será utilizado como parâmetro na seleção de indivíduos aptos a realizarem a geração de novas soluções. Posteriormente, o algoritmo entra em um laço de ng gerações voltadas à evolução e formação de novas populações. Em cada interação do ciclo, o algoritmo seleciona indivíduos baseando-se nos valores de aptidão. Em seguida, os indivíduos selecionados são encaminhados à recombinação, onde duas ou mais soluções terão o material genético intercalado para a geração de novos cromossomos. Essas novas soluções são encaminhadas à etapa de mutação, onde poderão sofrer modificações aleatórias em alguns de seus genes. Após as recombinações e mutações, o GA ainda pode utilizar algum procedimento de atualização da população, por exemplo, alguma estratégia elitista para encaminhar os melhores indivíduos para a próxima geração. Ao final, após ng gerações ou até alcançar um critério de parada, o algoritmo retornará o indivíduo que apresenta o melhor valor de aptidão.

Uma contextualização mais clara sobre os elementos típicos de um GA é apresentada nas próximas subseções.

Figura 5 – Fluxograma de um GA típico.



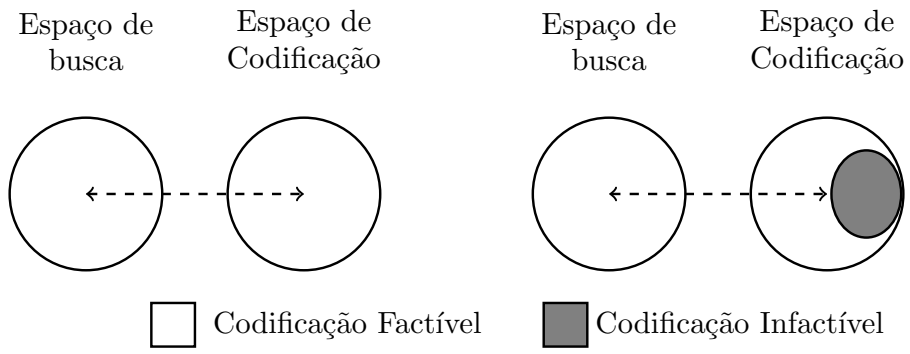
Fonte: adaptado de Kang et al. (2011).

2.2.1 Codificação

Um dos maiores desafios na aplicação de um GA é a sua representação. Esta etapa exige uma cuidadosa engenharia na abstração dos dados, isto é, uma caracterização que possa ser fiel à abstração do problema e que ao mesmo tempo, possa ser flexível aos mecanismos de um GA. Além de codificar o problema, a representação pode ter impacto no desempenho do algoritmo. Por exemplo, no problema das oito rainhas, é possível representar cada solução como uma cadeia binária ou como um agrupamento de números inteiros. Ambas representações podem ser aplicadas aos operados genéticos, todavia, podem apresentar comportamento diferente nas evoluções do algoritmo. Jelodar et al. (2006) acrescenta que a representação que o cromossomo utiliza é crucial para o desempenho do GA, definindo se o espaço de busca pode ser alcançado ou se cromossomos produzidos são válidos. Essa dualidade entre espaço de busca e codificações válidas e inválidas é ilustrada na Figura 6.

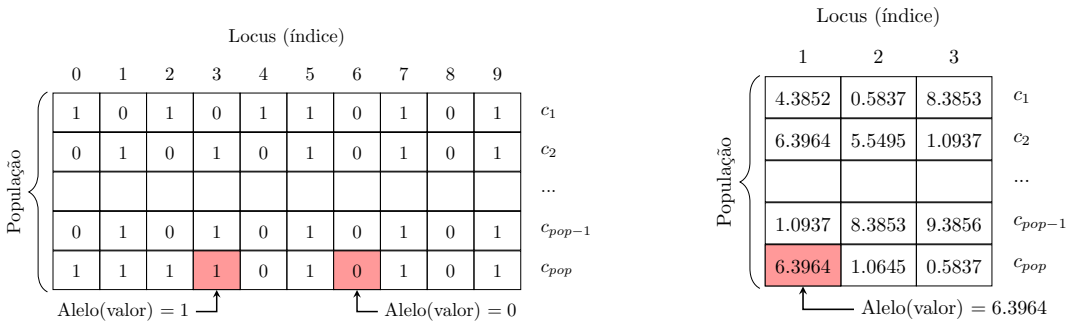
Na decodificação, esta estrutura é traduzida para uma solução do problema. Um agrupamento de diversas soluções caracteriza uma população, isto é, uma coleção de indivíduos de cada geração. Dessa forma, a população inicial pode ser gerada aleatoriamente ou baseada em regras, heurísticas e algoritmos (DHINGRA; GUPTA; BISWAS, 2014). A evolução de cada população é realizada pelos mecanismos de reprodução, onde cada ciclo

Figura 6 – Relação 1 para 1 entre espaço de busca e espaço de codificação.



Fonte: adaptado de Hou, Ansari e Ren (1994).

Figura 7 – Exemplos de algumas codificações empregadas por GAs.



(a) Codificação binária. (b) Codificação real.

Fonte: adaptado de Coello et al. (2007).

evolutivo representa uma geração (população atual do ciclo).

A Figura 7 ilustra dois modelos de codificação geralmente adotados em GAs: representação binária e representação com valores reais. Os genes formam uma coleção de símbolos que representam os cromossomos, onde cada fragmento é parte da solução codificada (GOLUB; KASAPOVIC, 2002). Os alelos representam os diferentes valores que um gene pode assumir, onde a delimitação de valores obedece a representação empregada – binária, real, dentre outras. Locus define a posição de um determinado gene dentro de um cromossomo (COELLO et al., 2007).

2.2.2 Otimização

Independente do cenário aplicado, os indivíduos de uma população necessitam de alguma medida que reflita seu desempenho como solução do problema. Neste caso, o

valor de aptidão é utilizado e é o principal índice relacionado à otimização provida pelo GA. Além disso, a aptidão é utilizada como critério classificatório nas operações de seleção. Em resumo, a aptidão é uma medida que determina o quão bom um cromossomo otimiza a função objetivo (JELODAR et al., 2006). Em problemas de minimização, soluções de maior aptidão são as de menor custo (GABRIEL; DELBEM, 2008). A medida de aptidão é recalculada sempre que um indivíduo é gerado ou modificado. Desse modo, otimizar uma função objetivo, dentro de um GA, é encontrar indivíduos com valor de aptidão ótimo ou próximo ao ideal no cenário explorado.

Uma otimização simples ainda pode ser expandida para otimizações biobjetivo e multiobjetivo. Esses objetivos geralmente são conflitantes, e encontrar uma solução para um único objetivo degrada a solução ao se considerar os demais objetivos (PILLAI et al., 2018). Variados esquemas são empregadas na tentativa de prover um equilibro entre os objetivos. A Expressão 8 apresenta o método da soma ponderada com duas funções objetivos arbitrárias para serem otimizadas (CHITRA; VENKATESH; RAJARAM, 2011). Nesta abordagem, as duas funções são combinadas a pesos que nivelam o nível de relevância pretendido a cada objetivo. Por exemplo, quando $\omega = 0$, somente a função f_2 é considerada. Em contrapartida, quando $\omega = 1$, somente a função f_1 é avaliada. Além disso, a soma de objetivos combinadas a pesos configura uma forma de conversão para um único objetivo.

$$\min f = \omega f_1 + (1 - \omega) f_2 \tag{8}$$

Mais semelhante aos problemas do mundo real, a otimização multiobjetivo geralmente procura determinar um balanceamento entre os objetivos, gerando um conjunto de soluções não dominadas, conhecido como soluções ótimas de Pareto (CHITRA et al., 2010). Desta forma, a otimização de um problema multiobjetivo pode ser modelada conforme a Expressão 9, onde N_{obj} define a quantidade de objetivos, f_i é a i-ésima função objetivo e x é uma cadeia de decisão que representa uma solução (CHITRA; VENKATESH; RAJARAM, 2011).

$$\min f(x) \quad i = 1, 2, ...N_{obj} \tag{9}$$

Segundo a definição de Chitra et al. (2010), em uma otimização multiobjetivo, entre duas soluções, x_1 e x_2, pode haver duas possibilidades: uma solução dominar a outra ou nenhuma dominar a outra. Desta forma, em um problema de minimização, uma solução x_1 domina uma solução x_2, se os seguintes critérios forem atendidos:

$$\begin{aligned} (i) \quad & \forall i \in \{1, 2, ...N_{obj}\} \rightarrow f_i(x_1) \leq f_i(x_2) \\ (ii) \quad & \exists j \in \{1, 2, ...N_{obj}\} \rightarrow f_i(x_1) \leq f_i(x_2) \end{aligned} \tag{10}$$

Conforme apresentado na Expressão 10, x_1 é uma solução não-dominada caso x_1 domine x_2. Se qualquer um dos critérios for violado, a solução x_1 não domina x_2. Portanto,

uma solução ótima de Pareto é a solução que, dentro do espaço de busca, não é dominada. Estas soluções formam o conjunto de soluções ótimas de Pareto ou fronteira ótima de Pareto.

2.2.3 Métodos de Seleção

O mecanismo de seleção é usado para selecionar indivíduos da população para futuras manipulações genéticas, por exemplo, recombinações e mutações (SINGH; PILLAI, 2014). Existem diversos mecanismos para selecionar cromossomos para as fases de reprodução, sendo os métodos de torneiro e roleta alguns exemplos de métodos populares de seleção. Na ausência de uma técnica específica, a seleção adota procedimentos totalmente aleatórios para definir os candidatos à reprodução. Independente do procedimento, os indivíduos selecionados são identificados como pais, e os novos indivíduos gerados são caracterizados como filhos ou descendentes. As seguintes subseções resumem o funcionamento de algumas técnicas geralmente empregadas.

2.2.3.1 Roleta

Segundo a definição de Singh e Pillai (2014), uma roleta é definida como a probabilidade cumulativa de seleção. Sendo assim, diversos intervalos que compõem a roleta são preenchidos por diversos cromossomos diferentes. O comprimento desses intervalos é proporcional a probabilidade de seleção do indivíduo. Esse preenchimento é definido conforme a Expressão 11. Após a construção da roleta, um número aleatório é gerado indicando qual intervalo (cromossomo) será selecionado. Naturalmente, indivíduos com maior valor de aptidão têm maior chance de serem selecionados. Entretanto, o procedimento não excluí indivíduos de baixo desempenho. O Algoritmo 1 sumariza as instruções para a execução da roleta. Em adição, a Figura 8 ilustra uma roleta formada por uma população com 6 indivíduos e seus respectivos valores de aptidão.

$$ProbSelect = \frac{f_i}{\sum_n f_i} \tag{11}$$

2.2.3.2 Torneio

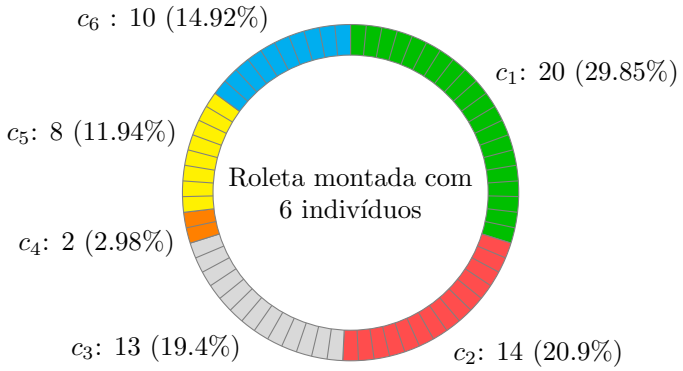
Com eficiência e uma implementação simples, o método do torneio é um dos procedimentos de seleção mais populares (GOLDBERG; DEB, 1991). Segundo a definição de Razali, Geraghty et al. (2011), na seleção por torneio, ts indivíduos da população são selecionados aleatoriamente. Em seguida, esses indivíduos pré-selecionados competem para definir o indivíduo selecionado. A disputa é uma competição pelos valores de aptidão mais altos. Naturalmente, o indivíduo com a maior aptidão vence o torneio e é encaminhado à reprodução. A Figura 9 ilustra um exemplo da aplicação do torneio em uma população com 6 indivíduos, onde ts identifica a quantidade de indivíduos pré-selecionados para o

Algoritmo 1 Algoritmo de Seleção por Roleta Proporcional – adaptado de Singh e Pillai (2014)

entrada: População;
saída: Indivíduo selecionado;

1: Calcule a probabilidade de seleção para todos os indivíduos na população usando a Expressão 11;
2: Calcule a probabilidade cumulativa para cada indivíduo usando: $cmProb(i) = cmProb(i-1) + ProbSelect(i)$
3: Gere um número aleatório r;
4: **se** $cmProb(i-1) < r \leq cmProb(i)$ **então**
5: Selecione um indivíduo i;
6: **fim se**
7: **retorne:** Indivíduo selecionado;

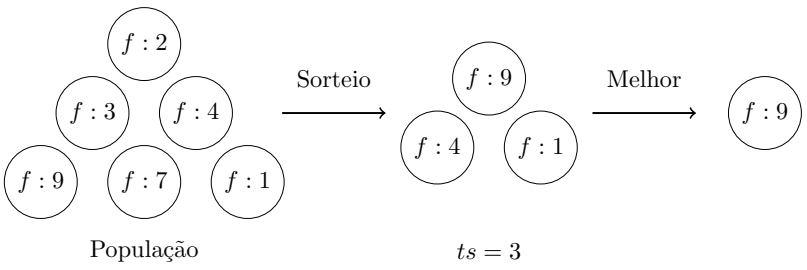
Figura 8 – Exemplo de Seleção por Roleta.



Fonte: adaptado de Singh e Pillai (2014).

torneio. Um torneio binário é uma instância do torneio tradicional, onde dois indivíduos são sorteados, e aquele com o maior valor de aptidão é selecionado para reprodução (AMIRJANOV; SOBOLEV, 2017).

Figura 9 – Exemplo de Seleção por Torneio.



Fonte: adaptado de Razali, Geraghty et al. (2011).

2.2.3.3 Classificação (ranking)

A seleção por classificação é implementada sobre um conceito simples: soluções dispostas na população são ordenadas da pior para a melhor através de seu desempenho na aptidão, onde, posteriormente, é designado uma probabilidade de seleção baseada nessa classificação (KANG; ZHANG; CHEN, 2011).

Grefenstette (2000) e Daoud e Kharma (2011) alertam que diferentemente da seleção proporcional, na qual os indivíduos com maior valor de aptidão podem dominar a população em poucas gerações uma vez que a probabilidade de seleção do cromossomo é proporcional a sua aptidão, a seleção baseada em classificação reduz a pressão seletiva por cromossomos com valor de aptidão superior ao assinalar a probabilidade de seleção de cada cromossomo baseando-se na classificação comparada aos demais indivíduos. Além disso, a seleção por ranqueamento mantêm a diversidade da população e auxilia na redução de convergência prematura da população (GREFENSTETTE, 2000).

2.2.3.4 Roleta com Classificação

Conforme o método tradicional da Roleta, uma estrutura é montada definindo intervalos que representam os indivíduos. Contudo, nesta técnica, os cromossomos são classificados (hierarquicamente) conforme seus valores de aptidão. Desse modo, o indivíduo com o melhor valor de aptidão é classificado como *pop*, enquanto que o pior indivíduo é classificado como 1 (SINGH; PILLAI, 2014). A Expressão 12 apresenta a atribuição de um novo valor de aptidão após a etapa de classificação, onde *sp* é a pressão seletiva, *pop* é o tamanho da população e $fit_{rank}(r)$ é o valor de aptidão baseado nas classificação dos cromossomos na posição r (RAZALI; GERAGHTY et al., 2011):

$$fit_{rank}(r) = 2 - sp + (2(sp-1)\frac{(r-1)}{(pop-1)}) \tag{12}$$

Conforme as definições apresentadas por Singh e Pillai (2014), o valor de pressão seletiva pode variar entre 1.0 e 2.0, e o valor de aptidão baseado em classificação é utilizado para calcular a probabilidade de seleção para cada indivíduo posicionado em r, conforme a Expressão 13.

$$prob\_select(r) = \frac{fit_{rank}(r)}{\sum popfit_{rank}(r)} \tag{13}$$

Após a definição dos valores de aptidão, baseados em classificação, a roleta pode ser montada conforme seu algoritmo tradicional (Subseção 2.2.3.1). As instruções para a Roleta com Classificação são enumeradas no Algoritmo 2.

Algoritmo 2 Algoritmo de Seleção de Roleta com Classificação – adaptado de Singh e Pillai (2014)

entrada: População;
saída: Indivíduo selecionado;

1: Defina classificações para todos os indivíduos de acordo com os valores de aptidão;
2: Calcule a classificação baseada na aptidão conforme a Expressão 12;
3: Calcule a probabilidade cumulativa para cada indivíduo usando:
$cmProb(i) = cmProb(i-1) + ProbSelect(i)$
4: Gere um número aleatório r;
5: **se** $cmProb(i-1) < r \leq cmProb(i)$ **então**
6: Selecione um indivíduo i;
7: **fim se**
8: **retorne:** Indivíduo selecionado;

2.2.4 Operadores de Reprodução

No contexto dos GAs, é necessário implementar mecanismos que explorem o espaço de busca atrás de outras soluções candidatas. Os mecanismo adotados são os operadores de recombinação e mutação, baseados nos conceitos biológicos de reprodução sexuada e assexuada, respectivamente.

2.2.4.1 Recombinação

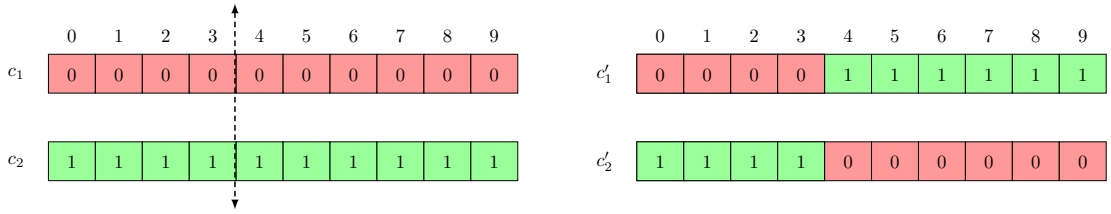
Conforme destacado no nome, a recombinação é um operador que mistura o material genético de um par de genótipos (pais) em um, dois ou mais genótipos descendentes (filhos), sendo a recombinação de ponto único o modelo mais simples adotado (ZOMAYA; WARD; MACEY, 1999). De maneira análoga ao operador de mutação, a recombinação é um operador estocástico: as escolhas de quais partes de cada pai serão combinadas e como isso será feito dependem de fatores aleatórios. A combinação de um número maior de pais também é possível sendo denominada como operadores de recombinação com alta aridade, todavia, sem um correspondente biológico (EIBEN; SMITH et al., 2003).

Um exemplo clássico de recombinação é ilustrado pela Figura 10, demonstrando a recombinação de ponto único adotada em duas cadeias binárias. Ao recombinar os genes através de um ponto de corte, é possível criar duas novas soluções, ambas construídas com informações cruzadas de ambos os pais. O ponto de corte é definido arbitrariamente, variando em cada aplicação do operador.

2.2.4.2 Mutação

A mutação é um operador unário que, ao ser aplicado em um indivíduo, gera um mutante modificado ou um novo descendente. Este tipo de operador é sempre estocástico, isto é, sua saída (o filho) dependerá de uma série de escolhas aleatórias, causando uma mudança não enviesada (EIBEN; SMITH et al., 2003). Em GAs, mutação é considerado

Figura 10 – Recombinação de ponto único aplicado em codificação binária.

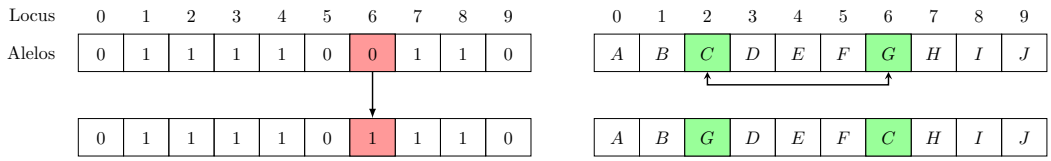


Fonte: do autor.

um método para recuperar material genético perdido (GOLUB; KASAPOVIC, 2002). Além disso, a mutação também permiti uma exploração mais global do espaço de busca. Introduzido por Holland (1975), a mutação surgiu como um operador para mudança ocasional dos bits dos indivíduos, atuando sob uma taxa probabilística de mutação $p_m \in [0, 1]$ por bit (BACK; FOGEL; MICHALEWICZ, 2000).

A Figura 11 apresenta dois exemplos distintos de mutação, em que cada um é aplicado em um tipo de representação específica: binária e baseada em permutação. Na cadeia binária, um locus sorteado tem seu respectivo gene modificado, de maneira semelhante ao modelo introduzido por Holland (1975). Em contrapartida, a cadeia baseada em permutação é formada por símbolos sem repetição. Logo, o procedimento adotado também efetua uma pequena modificação nos genes do indivíduo, todavia, sem produzir uma codificação inválida.

Figura 11 – Mutação aplicada em cromossomos binários e de permutação.



Fonte: do autor.

2.2.5 Migração

Ao final de cada geração, o tamanho final da população aumenta, uma vez que novos indivíduos foram formados como consequência do processo evolutivo. Portanto, é necessário um controle do tamanho dessa população. Em aspecto formal, dada uma população de tamanho *pop* e a quantidade de novos vizinhos, representada por *popn*, a próxima geração será denotada por $pop + popn$ possíveis combinações – representando estratégias de reaproveitamento e descarte de indivíduos – ou somente *pop* novos indivíduos. Assim, em conformidade com a definição de Pillai et al. (2018), uma estratégia elitista envolve

considerar uma pequena proporção dos indivíduos com o melhor valor de aptidão para nova população. Existem variadas estratégias elitistas, como a substituição, em que indivíduos com baixo desempenho são substituídos na próxima geração por indivíduos com alto valor de aptidão (PILLAI et al., 2018), e a condução, em que indivíduos com bom desempenho são encaminhados diretamente à próxima geração (XU et al., 2014).

2.3 Revisão Sistemática da Literatura

Revisão Sistemática da Literatura (SLR[6]) trata-se de interpretar e avaliar toda a pesquisa relevante para uma determinada questão de pesquisa, fenômeno ou tópico de alguma área (KITCHENHAM, 2004). SLR propõe a utilização de um método rigoroso, confiável e verificável para proporcionar uma avaliação clara de um tópico de pesquisa. Além disso, SLR é uma das maiores ferramentas usadas para agregar experiências vindas de uma coleção de diferentes estudos com o objetivo de responder uma questão de pesquisa (BUDGEN; BRERETON, 2006). Ao aplicar a SLR é possível identificar lacunas que possam propor novas investigações, sumarizar todas as evidências relativas a uma tecnologia ou técnica e fornecer um plano de fundo que possa proporcionar novas pesquisas. Outras aplicações também permitem reunir material para suporte de novas hipóteses e examinar evidências empíricas que suportem ou contradizem hipóteses (KITCHENHAM, 2004).

A maior vantagem da utilização da SLR é a capacidade de reunir informações suficientes para poder observar um efeito em um grande extensão de trabalhos, investigando suas configurações e seus métodos empíricos. As informações retornadas pela SLR podem ser utilizadas em ambos os casos (KITCHENHAM, 2004):

- **Informações consistentes:** demonstra que o evento provém de evidências do fenômeno;
- **Informações inconsistentes:** a fonte das variações pode ser estudada.

Em estudos quantitativos também é possível aplicar técnicas de meta-análise – sintetizar resultados quantitativos de diferentes trabalhos através de um procedimento estático– para combinar dados e, consequentemente, aumentar a probabilidade de encontrar fenômenos que pequenos estudos individuais não conseguem detectar. Todavia, isso pode ser uma desvantagem desde que abre a possibilidade para detecção de pequenos vieses como efeitos verdadeiros (KITCHENHAM, 2004). Qualquer estudo que possa contribuir com a SLR é denominado estudo primário, enquanto que uma SLR é considerada uma forma de estudo secundário (BUDGEN; BRERETON, 2006).

O conceito de revisão sistemática é baseado na ideia de evidências e foi desenvolvida inicialmente na medicina, formulada por Group et al. (1992), em que a opinião de

[6] Do inglês, *Systematic Literature Review*.

um médico especialista é tão confiável quanto a reunião de resultados sustentados por experimentos científicos (KITCHENHAM et al., 2009).

Os procedimentos adotados pela SLR são flexíveis o bastante para serem adotados em variados cenários. Desse modo, alguns exemplos da aplicação são:

- ❏ Tecnologias de automação (TSAFNAT et al., 2014);
- ❏ Percepção semiótica de interfaces de usuário (ISLAM, 2013);
- ❏ Coleta de dados por telefones celulares (SAHIN; YAN, 2013);
- ❏ Estratégias de inferência (WOODALL; BRERETON, 2010);
- ❏ Filtragem de spam (CORMACK, 2006);
- ❏ Técnicas de aprendizado de máquina para previsão de falhas de software (MALHOTRA, 2015);
- ❏ Teste de software científico (KANEWALA; BIEMAN, 2014);
- ❏ Engenharia de software para jogos de computador (AMPATZOGLOU; STAMELOS, 2010);
- ❏ Táticas arquitetônicas para cyber-forrageamento (LEWIS; LAGO, 2015);
- ❏ Agentes aplicados na saúde (ISERN; MORENO, 2016).

Uma SLR implementa uma estratégia pré-definida de busca por trabalhos que permita uma avaliação geral de todos os trabalhos retornados. A estratégia deve permitir que os pesquisadores possam identificar e reportar trabalhos que apoiem sua hipótese de pesquisa, assim como estudos que não a suportem. Considerando que a maioria das pesquisas começa com uma revisão bibliográfica, e que para a utilização desta revisão, a mesma deve ser clara e minuciosa, a aplicação de uma SLR configura uma boa opção de revisão, uma vez que SLR pode sintetizar trabalhos existentes obedecendo critérios de organização. Todavia, uma revisão clássica da bibliografia (método *ad hoc*) ainda é importante, pois pode ser utilizada para determinar as palavras-chaves utilizadas na SLR, assim como expandir o conhecimento dos pesquisadores sobre o tema.

A execução da SLR deve ser separada em fases que facilitem a execução e avaliação dos estudos. Essas fases devem documentar as ações tomadas pelos pesquisadores com o intuito de facilitar futuras re-avaliações. Uma possível divisão é documentada por Biolchini et al. (2005) e adaptada às fases a seguir:

- ❏ **Concepção:** identificar os conceitos voltados ao problema do estudo e obter os estudos que levem às evidências;

- **Comparação:** estudar e comparar os estudos minunciosamente, a procura de novas evidências;
- **Reunião:** sintetizar os resultados encontrados e gerar novas conclusões.

Budgen e Brereton (2006) sugerem uma divisão baseada em planejamento, condução e reporte da revisão, ambas contextualizadas nas próximas subseções e baseando-se no guia disponibilizado por Kitchenham (2004).

2.3.1 Planejamento da Revisão

Inicialmente, o planejamento da revisão contempla a necessidade para uma SLR, isto é, para que todas as informações sejam sumarizadas, é necessário determinar o fenômeno que será investigado (KITCHENHAM, 2004). As seguintes questões são alguns exemplos utilizados para assegurar a necessidade da revisão:

- Quais são os objetivos da revisão?
- Quais critérios de inclusão e exclusão serão adotados?
- Quais serão as fontes analisadas?
- Como os dados serão extraídos dos estudos?
- Como as informações dos estudos serão combinadas?

A segunda parte desta etapa caracteriza a elaboração do **Protocolo de Revisão**. Este protocolo é um documento que descreve em detalhamento todos os procedimentos adotados pela SLR. Deve ser estruturado com o objetivo de permitir uma ampla visualização das etapas adotadas e uma futura reprodução do trabalho executado. Protocolos podem ser revisados por outros pesquisadores ou por orientadores de pesquisa (no caso de elaboração de teses ou dissertações). O documento pode conter outras informações adicionais como:

- As questões de pesquisa que devem ser respondidas ao final do trabalho;
- Razões para a adoção da revisão e contextualização do tema;
- Estratégias que serão utilizadas no levantamento de estudos;
- Informações de recursos de busca, base de dados e identificação de jornais e conferências nas fontes selecionadas;
- Procedimentos e critérios empregados para inclusão e exclusão de estudos;
- Procedimentos para avaliação da qualidade dos trabalhos selecionados;

- ❏ Estratégias para extração e síntese dos dados presentes nos trabalhos;
- ❏ Cronograma do projeto.

A mais importante atividade na definição do protocolo é a elaboração da questão de pesquisa. Ao elaborá-la é necessário observar quais são os alvos que poderão ser beneficiados ao encontro da resposta investigada. A construção de uma questão de pesquisa deve ser orientada sobre três pontos de vista, seguindo a orientação dos guias médicos de SLR:

- ❏ A população, por exemplo, o que será afetado pelas intervenções;
- ❏ As intervenções, isto é, as possíveis técnicas que serão comparadas;
- ❏ Os resultados originados de medidas que demonstrem a diferença entre as intervenções.

A partir da caracterização da questão é possível dividi-la em outras questões secundárias voltadas a reforçar a importância do questão principal e estender o impacto dos possíveis resultados.

2.3.2 Condução da Revisão

A identificação da pesquisa é primeira fase da condução e trata-se de definir estratégias para encontrar a maior quantidade de estudos sobre a questão de pesquisa sem adotar um procedimento enviesado, isto é, não levantar trabalhos somente sobre as hipóteses em que o pesquisador mais acredita, mas sim, obter trabalhos associados ao tema investigado sem seguir qualquer tendência (KITCHENHAM, 2004).

Inicialmente, uma revisão preliminar (*ad hoc* ou Narrativa) em SLRs aplicadas em estudos primários relevantes pode identificar os termos comuns ao assunto investigado. Além das palavras comuns ao tema, é possível identificar sinônimos que podem estender o alcance das buscas posteriores. Tentar executar buscas com os campos observados, revisar os trabalhos retornados e consultar especialistas são outros exemplos de estratégias iniciais.

Ao identificar os termos de busca utilizados, deve-se combiná-los para a construção de chaves de busca mais robustas. Os termos podem ser combinados com operadores lógicos (**OR** ou **AND**) para flexibilizar a busca por estudos alvo da questão de pesquisa. Algumas outras táticas para levantar estudos são a utilização da chave de busca em bases eletrônicas, análise da lista de referência dos trabalhos mais relevantes retornados, pesquisa em jornais específicos ou conferências, busca por pesquisadores específicos e estender as buscas por outros mecanismos da Internet.

Após a obtenção dos estudos, é recomendável que o material seja mapeado em ferramentas de gerenciamento bibliográfico. Além de armazenar suas informações básicas

(base eletrônica, jornal, etc), é necessário que a RS documente qualquer mudança efetuada na base através de relatórios. As mudanças devem ser justificadas e descritas com o intuito de proporcionar uma nova análise. Esses procedimentos proporcionam que a SLR seja um serviço transparente e replicável.

A fase de seleção de estudos primários caracteriza-se por utilizar a questão de pesquisa como base para a elaboração de critérios de inclusão e exclusão (KITCHENHAM; BRERETON, 2013). Esses critérios devem ser definidos de maneira a classificar os estudos corretamente. O processo de seleção deve ser documentado com o registro de inclusão e exclusão dos trabalhos analisados. Para a execução da seleção, grupos de pesquisa devem estar sincronizados e cada desacordo deve ser discutido e resolvido. Quando a RS é executada por um único pesquisador, trabalhos selecionados devem ser discutidos com algum especialista da área. Trabalhos em que a seleção é aplicada em meio a incertezas, podem ser complementados com uma análise sensitiva.

A avaliação da qualidade dos estudos reforça os critérios de inclusão e exclusão, o que caracteriza a terceira fase da condução. Khan et al. (2001) sugerem que a avaliação de qualidade está relacionada com a validade dos estudos e a diminuição de viés bibliográfico. A validade interna caracteriza o estudo apropriado para prevenir erros sistemáticos. Já a validade externa está relacionada com trabalhos que possuem efeitos aplicáveis fora do estudo.

A extração de dados é uma fase voltada a obter informações dos estudos necessários às respostas das questões de pesquisa e voltados aos critérios de qualidade do estudo (KITCHENHAM et al., 2009). É necessário definir os dados que serão extraídos juntamente com seus formatos, por exemplo, o formato de informação numérica é um pré-requisito para técnicas de meta-análise. Para que os dados extraídos possam ser analisados e organizados, pode-se utilizar ferramentas como formulários eletrônicos onde cada estudo terá além dos dados extraídos, informações básicas como nome, data da extração, título, detalhes da publicação e espaço para notas. Para estender a abrangência do estudo, a SLR também pode obter dados ainda não publicados – através da comunicação e autorização dos autores, incompletos na publicação ou dados que requerem algum tipo de manipulação – por exemplo, análise sensitiva.

A síntese dos dados caracteriza a fase de agrupamento e sumarização dos resultados encontrados nos estudos selecionados (KITCHENHAM, 2004). A síntese pode ser construída para resultados descritivos, isto é, não qualitativos. Todavia, a construção de síntese de resultados quantitativos também é possível através de técnicas estatísticas de meta-análise. Os dados extraídos devem ser apresentados de maneira a facilitar a visualização de suas similaridades e diferenças, seja por tabelas ou gráficos.

2.3.3 Reporte da Revisão

Os resultados encontrados através das fases anteriores podem ser contextualizados usualmente em teses ou em artigos científicos (*papers*). Geralmente, um jornal atribui restrições quanto ao tamanho e organização dos documentos. Neste caso, quando as restrições não permitem apresentar todo o processo sistemático em grande detalhamento, recomenda-se que o artigo referencie documentos técnicos que completem todos os passos executados no processo. Jornais também podem exigir um rigoroso processo de revisão através de seus colaboradores. Esse processo contribui com a SLR uma vez que caracteriza uma revisão adicional por especialistas da área pesquisada. Uma SLR que apresente resultados práticos pode estender sua publicação para documentos não técnicos como revistas e páginas da Web (KITCHENHAM, 2004).

Capítulo 3

Revisão Sistemática sobre GAs aplicados ao MTSP

Neste capítulo as principais informações referentes ao planejamento, condução e reporte da revisão sistemática são reunidas. Desse modo, as seções desse capítulo apresentam todos os elementos necessários à reprodução da revisão executada.

Inicialmente, a Seção 3.1 contextualiza todos os elementos empregados no protocolo de revisão. Esses elementos representam as questões de pesquisas, regras da revisão e demais procedimentos de pesquisa. Em seguida, com a realização de uma análise da coleção de trabalhos obtida, a Seção 3.2 apresenta respostas às questões de pesquisa levantadas no protocolo de revisão.

3.1 Protocolo de Revisão Sistemática

Esta seção apresenta o protocolo aplicado nas fases da revisão. A documentação foi elaborada em conformidade aos conceitos apresentados no capítulo 2, Seção 2.3, de modo que sua estrutura permite uma visualização das etapas adotas e, inclusive, a reprodução de seus resultados.

Inicialmente, a Subseção 3.1.1 sumariza as questões de pesquisa essenciais à execução dessa revisão. Em seguida, os termos iniciais e finais selecionados na construção da chave de busca são listados na Subseção 3.1.2. A chave de busca utilizada na recuperação de estudos é apresentada na Subseção 3.1.3. A Subseção 3.1.4 lista as bases de dados consultadas. Regras para a inclusão e exclusão de estudos na revisão são contextualizadas na Subseção 3.1.5. Por fim, cada fase na execução da revisão é definida na Subseção 3.1.6.

3.1.1 Questões de Pesquisa

Considerando o contexto desta monografia, isto é, algoritmos genéticos aplicados ao problema de escalonamento de tarefas em multiprocessadores, uma questão principal de

pesquisa (QP) foi desenvolvida para guiar a investigação dessa revisão, sendo essa:

> **QP:**"Entre o período de 1990 a 2018, quais são os estudos centrais sobre algoritmos genéticos aplicados ao problema de escalonamento estático de tarefas em multiprocessadores?"

Com o objetivo de estender os possíveis resultados desse estudo, o protocolo ainda define as seguintes questões secundárias (QS):

- **QS.01:** Quais são as codificações genéticas utilizadas? Além disso, quais são as recombinações, mutações e formas de seleção empregadas nestas representações?
- **QS.02:** Qual o período com a maior quantidade de trabalhos?
- **QS.03:** Quais são os cenários considerados nesses trabalhos?
- **QS.04:** Como esses trabalhos lidam com o custo de comunicação?
- **QS.05:** Quais são as principais funções objetivo consideradas?
- **QS.06:** Quais desses trabalhos possuem maior fator de impacto e Eigenfactor?
- **QS.07:** Quais desses trabalhos possuem melhor classificação no extrato Qualis?
- **QS.08:** Quais desses trabalhos possuem a maior quantidade de citações?
- **QS.09:** Dentre os trabalhos levantados, quais são os mais citados dentro da coleção?
- **QS.10:** Quais regiões do planeta têm a maior quantidade de autores?
- **QS.11:** Quais são as expressões mais utilizadas nos resumos e nas palavras-chave?

3.1.2 Palavras-chave e Sinônimos

Inicialmente, é necessário definir quais são os termos principais utilizados na formação das chave de busca. É essencial que essas palavras consigam alcançar os índices utilizados pelas bases digitais no armazenamento e organização dos estudos publicados. Portanto, com base em uma revisão informal da literatura, as primeiras palavras-chave e seus respectivos sinônimos selecionados foram:

- *Genetic algorithms*; *genetic operators*;
- *Directed acyclic graph*; DAG;
- *Multiprocessor scheduling*;
- *Scheduling*.

Os termos supracitados têm forte generalização, isto é, alcançam uma quantidade de estudos relativamente alta. Todavia, ao considerar os variados mecanismos de indexação utilizados pelas bases digitais e a possibilidade de uma grande quantidade de trabalhos fora do tema serem resgatados, foi necessário refinar as palavras utilizadas. Os novos termos adotados restringem a quantidade de trabalhos, porém, por serem mais específicos, obtêm estudos mais relacionados ao tema do trabalho. Dessa forma, as novas palavras-chave adotadas bem como seus respectivos sinônimos foram:

- *evolutionary algorithm*; *genetic algorithm*;
- *task scheduling*;
- *parallel*; *multiprocessor*;
- *directed acyclic graph*; DAG; *workload*;
- *representation*; *encoding*;
- *genetic operators*;
- *objective function.*

3.1.3 Chave de Busca

Para execução da SLR, uma chave de busca foi construída com base nas questões de pesquisa da Seção 3.1.1 e nos termos selecionados na Seção 3.1.2. Cada expressão presente na chave é formada por palavras-chave que representam mecanismos de solução ou aspectos do problema. Além disso, a chave emprega os operadores lógicos **OR** e **AND** para eliminar sinônimos e/ou combinar palavras-chave com diferentes semânticas em uma única expressão de pesquisa. Ao final, a chave de busca é inserida em mecanismos de busca e indexação presentes nas bases digitais (Seção 3.1.4) sem qualquer tipo de modificação.

Diversos experimentos foram realizados na tentativa de moldar uma chave de busca mais adequada e direcionada. Por exemplo, a Tabela 2 apresenta a primeira chave de busca construída sobre as palavras-chave definidas nas fases iniciais da pesquisa. Conforme citado na Seção 3.1.2, os termos iniciais utilizados retornavam uma grande quantidade de trabalhos fracamente relacionados com o tema. Desta forma, utilizando as palavras-chave mais adequadas, a Tabela 3 apresenta a versão final da chave de busca empregada na SLR deste trabalho. Cabe ressaltar que, apesar do refinamento na construção da chave, o emprego desta resulta no retorno de diversos trabalhos. Consequentemente, nas fases posteriores, é aplicado um processo de filtragem de modo a remover trabalhos não relacionados.

Tabela 2 – Primeira chave de busca construída.

("*genetic algorithms*" **OR** "*genetic operators*") **AND** ("*directed acyclic graph*" **OR** DAG) **AND** "*multiprocessor scheduling*" **AND** *Scheduling*

Tabela 3 – Chave de busca final aplicada na SLR.

("*evolutionary algorithm*" **OR** "*genetic algorithm*") **AND** "*task scheduling*" **AND** (*parallel* **OR** *multiprocessor*) **AND** ("*directed acyclic graph*" **OR** DAG **OR** *workflow*) **AND** (*representation* **OR** *encoding*) **AND** "*genetic operators*" **AND** "*objective function*"

3.1.4 Bases de Dados

A Seção 3.1.3 apresentou a chave de busca (Tabela 3) montada para pesquisa e recuperação de trabalhos disponíveis na Web. Dessa forma, a Tabela 4 lista as bases de dados utilizadas para consulta de estudos. Os repositórios foram escolhidos com base em revisões informais da literatura e em seu nível de abrangência. Além das bases indicadas, o protocolo também considerou a base ACM Digital Library; entretanto, devido particularidades internas do seu mecanismo de busca, os estudos resultantes não foram satisfatórios para a execução do projeto. A mesma chave de busca foi aplicada em todas as bases selecionadas, isto é, sem quaisquer modificações em sua estrutura.

Tabela 4 – Bases de dados utilizadas na busca.

Base de Dados	**Endereço Virtual**
Google Scholar	<scholar.google.com>
IEEE Xplore Digital Library	<ieeexplore.ieee.org>
Elsevier-Science Direct	<sciencedirect.com>
Portal de Periódicos Capes	<periodicos.capes.gov.br>

3.1.5 Critérios de Inclusão e Exclusão

Devido à natureza interna dos mecanismos de pesquisa, a aplicação de uma chave de busca pode recuperar uma vasta quantidade de estudos não relacionados diretamente ao tema desse estudo. Mesmo com uma chave composta por termos fortemente relacionados, os algoritmos de busca podem assimilar fragmentos da chave a outros índices dispostos na base. Desse modo, o protocolo define os seguintes critérios de inclusão e exclusão:

1. Trabalhos devem abordar o MTSP estático sem o tratamento de *deadlines*, isto é, não há um limite de tempo para execução e todas as informações referentes ao problema são conhecidas antes da realização do escalonamento;

2. Trabalhos devem empregar GAs ou EAs na resolução do MTSP, isto é, GAs ou EAs não são utilizados somente como um processamento externo;

3. Com exceção de heurísticas, trabalhos não podem combinar GAs ou EAs com outras meta-heurísticas;

4. Trabalhos devem estar escritos em inglês;

5. Trabalhos devem estar disponíveis para consulta via plataformas Web.

O primeiro critério adotado define as características do problema estudado. Dessa forma, outras variações do MTSP não são abordadas, o que reduz a quantidade de material analisado e define um direcionamento mais claro dos elementos pesquisados. O segundo critério também reduz o escopo da pesquisa ao determinar o uso de GAs e EAs na resolução do MTSP. De forma análoga, o terceiro critério exclui resoluções onde os GAs ou EAs são combinados com outras meta-heurísticas, uma vez que essa pesquisa aborda GAs ou EAs em suas versões mais básicas, isto é, as fases evolutivas dos algoritmos tendem a seguir padrões de ordem e implementação. Por fim, o quarto e o quinto critério foram adotados para elevar o nível de disponibilidade desses trabalhos, ou seja, para futuras consultas, os trabalhos podem ser facilmente acessados e estão escritos em uma linguagem amplamente utilizada.

Cabe ressaltar que as regras deste protocolo são uma extensão do modelo apresentado em (SILVA; GABRIEL, 2019), no qual um critério de exclusão foi incluído: trabalhos deviam ser publicados em periódicos. Esta restrição foi levantada com o objetivo de analisar trabalhos que pudessem ser avaliados por métricas de publicação. Logo, os resultados apresentados nesta monografia se estendem aos trabalhos publicados tanto em periódicos quanto em conferências.

3.1.6 Fases de Execução

Com a definição de todos os elementos supracitados, a SLR obedeceu às seguintes fases de execução:

- ❑ **Pesquisa por mecanismo de busca da Web:** Utilizando a chave de busca, as bases de dados são exploradas. Nessa fase, todas as referências retornadas na busca são catalogadas e armazenadas. Portanto, trabalhos com pouca relação com este estudo são mantidos para análises posteriores;

- ❑ **Filtragem de resultados:** Uma análise é executada sobre a coleção de referências recuperadas na fase anterior. A verificação busca identificar conformidades na definição do problema ou no detalhamento das técnicas. Nessa fase, estudos que não alcançam um ou mais critérios de inclusão/exclusão são descartados;

- ❑ **Pesquisa manual:** De posse de uma coleção inicial de estudos, a última fase do levantamento corresponde a uma verificação das referências bibliográficos de cada trabalho selecionado. Dessa forma, é possível localizar estudos que possam não ter sido recuperados nos mecanismos de busca. Além disso, nessa fase, as novas referências extraídas também passam pela análise dos critérios de inclusão/exclusão.

Os processos e documentos gerados nas fases supracitadas são inteiramente documentados por meio de planilhas e anotações. Esse histórico permite uma rápida consulta na tentativa de identificar as decisões tomadas em cada estudo revisado. Portanto, a documentação é um registro importante para demonstrar que todas as referências coletadas foram analisadas e que a SLR não foi executada sobre algum tipo viés.

3.2 Análise da Coleção

Esta seção[1] sintetiza as respostas referentes às questões de pesquisa da Subseção 3.1.1. A **QS.01** aborda aspectos referentes às codificações e operadores utilizados nas soluções. A resposta dessa questão é apresentada por meio do Capítulo 4. De forma análoga, **QS.05**, uma questão sobre as funções objetivo, é respondida por meio do Capítulo 5. Entretanto, as respostadas de ambas as questões, **QS.01** e **QS.05** são complementadas nesta seção.

Primeiramente, a Subseção 3.2.1 apresenta os trabalhos pesquisados juntamente a uma distribuição temporal de publicações. Aspectos dos cenários de aplicação são relacionados na Subseção 3.2.2. A Subseção 3.2.3 apresenta as métricas empregadas ao MTSP com GAs. Classificações das publicações da coleção são apresentadas na Subseção 3.2.4. A Subseção 3.2.5 contabiliza e ilustra a distribuição de pesquisadores globalmente. Os principais termos utilizados em palavras-chave e nos resumos são quantificados na Subseção 3.2.6. Um levantamento das principais técnicas de seleção é apresentado na Subseção 3.2.7. Da mesma forma, estratégias elitistas são levantadas na Subseção 3.2.8.

3.2.1 Coleção e Linha do Tempo

Ao executar a primeira fase da pequisa, um total de 430 trabalhos foram levantados nas bases consultadas: Google Scholar[2] (237), Elsevier-Science Direct[3] (157), IEEE Xplore Digital Library[4] (2) e Portal de Periódicos Capes[5] (34). Contudo, 50 desses trabalhos eram resultados duplicados entre as bases de dados. Portanto, ao final da consulta via chave de

[1] Os resultados apresentados nesta seção são uma extensão dos dados apresentados em (SILVA; GABRIEL, 2019), onde um filtro foi estabelecido ao apresentar somente trabalhos publicados em periódicos.

[2] Busca executada em 23 de Abril de 2018, às 22:58:04.

[3] Busca executada em 17 de Abril de 2018, às 10:32:47.

[4] Busca executada em 26 de Abril de 2018, às 16:25:42.

[5] Busca executada em 27 de Abril de 2018, às 17:25:36

busca, 380 trabalhos foram separados. Em seguida, a partir dessa coleção, 120 trabalhos foram pré-selecionados por meio da leitura das seções de resumo. Todos esses resultados, sejam descartados ou selecionados, foram documentados através de planilhas digitais. Na segunda fase, a filtragem considerou seções referentes aos aspectos das técnicas e a definição do problema. Sendo assim, a coleção inicial foi formada com 36 trabalhos. Na execução da terceira fase, ao analisar as referências bibliográficas da coleção inicial, 29 novos trabalhos encontrados, totalizando uma coleção com 65 trabalhos. O objetivo da terceira fase foi buscar trabalhos que não foram localizados com a chave de pesquisa e os mecanismos das bases digitais. Ao final, após uma análise aprofundada dos trabalho coletados, duas referências foram removidas por conflitarem com os critérios de inclusão e exclusão. Portanto, a coleção possui 63 trabalhos.

A coleção de trabalhos obtida na SLR é sumarizada cronologicamente na Tabela 5. Esse primeiro levantamento aborda a questão principal da pesquisa (**QP**), referente aos estudos centrais. Com a definição do intervalo de publicação entre 1990 e 2018, o primeiro trabalho encontrado sobre o tema foi o de Hou, Hong e Ansari (1990).

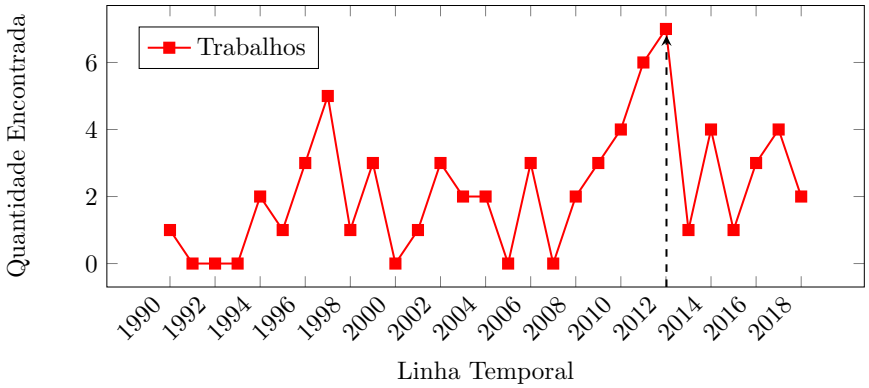
Tabela 5 – Coleção de trabalhos levantados.

Id	Artigo	Id	Artigo
1	Hou, Hong e Ansari (1990)	33	Gupta, Kumar e Agarwal (2010)
2	Benten e Sait (1994)	34	Chitra et al. (2010)
3	Hou, Ansari e Ren (1994)	35	Kaur, Chhabra e Singh (2010b)
4	Wang e Korfhage (1995)	36	Mohamed e Awadalla (2011)
5	Tsujimura e Gen (1996)	37	Wang et al. (2011)
6	Singh e Youssef (1996)	38	Kang, Zhang e Chen (2011)
7	Wang, Siegel e Roychowdhury (1996)	39	Sathappan et al. (2011)
8	Kwok e Ahmad (1997)	40	Chitra, Venkatesh e Rajaram (2011)
9	Tsuchiya, Osada e Kikuno (1997)	41	Daoud e Kharma (2011)
10	Aguilar e Gelenbe (1997)	42	Ahmad, Munir e Nisar (2012)
11	Woo et al. (1997)	43	Kaur e Singh (2012)
12	Wang et al. (1997)	44	Amalarethinam e Selvi (2012)
13	Tsuchiya, Osada e Kikuno (1998)	45	Xu et al. (2012)
14	Jezic et al. (1999)	46	Singh e Singh (2012)
15	Zomaya, Ward e Macey (1999)	47	Singh (2012)
16	Corrêa, Ferreira e Rebreyend (1999)	48	Panwar, Lal e Singh (2012)
17	Golub e Kasapovic (2002)	49	Awadall, Ahmad e Al-Busaidi (2013)
18	Liu, Li e Yu (2002)	50	Xu et al. (2014)
19	Topcuoglu e Sevilmis (2002)	51	Dhingra, Gupta e Biswas (2014)
20	Lee e Chen (2003)	52	Singh e Pillai (2014)
21	Zhong e Yang (2003)	53	Guzek et al. (2014)
22	Yao, You e Li (2004)	54	Hassan et al. (2015)
23	Wu et al. (2004)	55	Sheikh, Ahmad e Fan (2016)
24	Jelodar et al. (2006)	56	Morady e Dal (2016)
25	Demiroz e Topcuoglu (2006)	57	Ahmad et al. (2016)
26	Ramachandra e Elmaghraby (2006)	58	Akbari, Rashidi e Alizadeh (2017)
27	Hwang, Gen e Katayama (2008)	59	Amirjanov e Sobolev (2017)
28	Azghadi et al. (2008)	60	Gandhi, Nitin e Alam (2017)
29	Omara e Arafa (2009)	61	Sheikh, Ahmad e Arshad (2017)
30	Pop, Dobre e Cristea (2009)	62	Akbari (2018)
31	Bonyadi e Moghaddam (2009)	63	Pillai et al. (2018)
32	Kaur, Chhabra e Singh (2010a)		

Um fluxo temporal de publicações é ilustrado na Figura 12. É possível observar um aumento na quantidade de publicações após 1994, com o surgimento de estudos como (HOU; ANSARI; REN, 1994) e (BENTEN; SAIT, 1994). Respondendo a **QS.02**, 2011 a

2012 é o período com a maior quantidade de publicações. Além disso, é possível observar um aumento significativo na exploração de soluções ao MTSP a partir de 2010.

Figura 12 – Relação entre trabalhos e período de publicação.



Fonte: do autor.

3.2.2 Ambientes e Custo de Comunicação

Existem variadas características para montar um cenário do MTSP. Por exemplo, as tarefas podem ser distribuídas em processadores homogêneos, onde o custo computacional é o mesmo para cada tarefa, ou mesmo processadores heterogêneos, onde o custo computacional varia entre as diferentes unidades de processamento. Além disso, um cenário do MTSP pode optar pelo tratamento do envio de dados entre tarefas relacionadas, representando assim o custo de comunicação. Desta forma, as questões **QS.03** e **QS.04** são abordadas através dos gráficos da Figura 13.

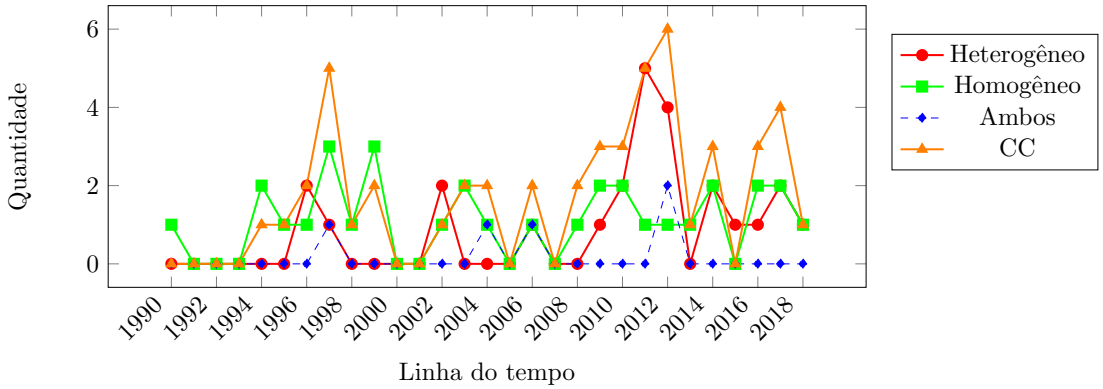
O cenário homogêneo é o mais predominante, sendo abordado em 32 trabalhos da coleção ($\approx 50,79\%$), com uma média de publicação de 1.10 trabalho por ano. Na sequência, cenários heterogêneos são abordados em 25 trabalhos da coleção ($\approx 39,68\%$) e possuem uma média de publicação de 0.86 trabalho por ano, sendo mais explorados na última década. Finalmente, cinco trabalhos da coleção abordaram ambos os cenários, sendo esses: Woo et al. (1997), Wu et al. (2004), Jelodar et al. (2006), Singh (2012), Panwar, Lal e Singh (2012).

O tratamento do custo de comunicação (CC) é outro elemento predominante nas publicações, estando presente em 50 trabalhos da coleção ($\approx 79,36\%$) e com uma média de aplicação de 1.72 trabalho por ano.

3.2.3 Métricas de Desempenho

Funções objetivo empregadas nas soluções para o MTSP são implementadas com o apoio de diversas métricas de desempenho. Nesse contexto, a Tabela 6 relaciona as

Figura 13 – Arquiteturas e custo de comunicação sobre a linha do tempo.



Fonte: do autor.

métricas e trabalhos levantados na coleção. Os trabalhos estão referenciados pelos seus respectivos IDs na Tabela 5. Para responder a **QS.05**, a relação apresentada na Tabela 6 é complementada pela sumarização das funções objetivo no Capítulo 5.

Estando presente em 62 trabalhos da coleção, a minimização do makespan é a métrica mais utilizada. Além disso, também é possível observar o crescimento de métricas relacionadas à sustentabilidade, por exemplo, a minimização da temperatura do sistema multiprocessador, representada por Sheikh, Ahmad e Fan (2016), Sheikh, Ahmad e Arshad (2017), e a minimização do consumo elétrico total, presente em Guzek et al. (2014), Sheikh, Ahmad e Fan (2016), Sheikh, Ahmad e Arshad (2017), Pillai et al. (2018). Isso se deve à crescente utilização comercial de sistemas distribuídos, notadamente por meio de soluções envolvendo nuvens computacionais (GARG; BUYYA, 2012; JAIN et al., 2013; ZHU et al., 2015).

Tabela 6 – Métricas de desempenho e trabalhos levantados.

Métricas de desempenho	**IDs dos trabalhos**
Minimizar makespan	1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62 e 63
Minimizar consumo elétrico	53, 55, 61 e 63
Maximizar confiabilidade do sistema	34, 37, 39 e 40
Minimizar flowtime	32, 34, 35 e 51
Maximizar balanceamento de carga	10, 29, 30 e 57
Minimizar temperatura do sistema	55 e 61
Maximizar paralelismo	58 e 62
Minimizar quantidade de processadores	19 e 25
Minimizar custo de comunicação	10
Maximizar utilização de recursos	44

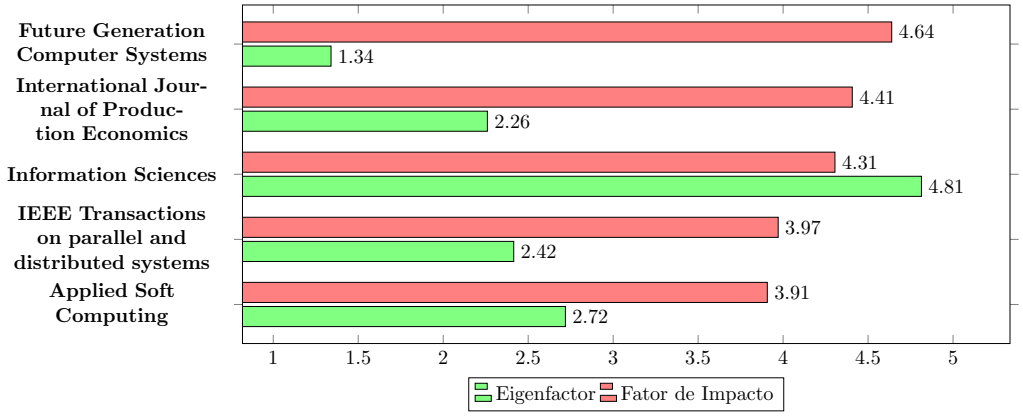
3.2.4 Métricas Avaliativas das Publicações

Existem variadas formas de avaliar a importância de uma publicação, tais como pontuação do periódico ou mesmo a quantidade de citações realizadas. Nesta dissertação optou-se por analisar dois índices comumente utilizados: o fator de impacto[6] e o índice Eigenfactor[7]. A medição da média de citações de um periódico é efetuada pelo fator de impacto, enquanto que a quantificação baseada na importância dos periódicos é realizada pelo índice Eigenfactor.

O gráfico da Figura 14 apresenta uma classificação de periódicos com base no fator de impacto (IF). Adicionalmente, com conversão de ordem de magnitude, o índice Eigenfactor (EF) também é ilustrado. Utilizando os dados presentes no gráfico, é possível responder à **QS.06**.

O periódico *Future Generation Computer Systems* obteve o fator de impacto mais alto, possuindo $IF = 4.639$ pontos, onde o trabalho de Wang et al. (2011) foi publicado. Em seguida, vem o periódico *International Journal of Production Economics*, com um $IF = 4.407$ pontos, publicando o trabalho de Ramachandra e Elmaghraby (2006). Apesar de estar classificado em quarto lugar, o periódico *IEEE Transactions on Parallel and Distributed Systems* possui cinco artigos na coleção, sendo essa a maior quantidade de trabalhos citados por um periódico neste projeto (HOU; ANSARI; REN, 1994), (CORRÊA; FERREIRA; REBREYEND, 1999), (ZOMAYA; WARD; MACEY, 1999), (WU et al., 2004) e (SHEIKH; AHMAD; FAN, 2016). Ao observar a classificação pelo índice Eigenfactor, *Information Sciences* é o periódico mais relevante, com $EF = 4.81$ pontos. Os trabalhos de Aguilar e Gelenbe (1997) e Xu et al. (2014) foram publicados nesse periódico.

Figura 14 – Fator de Impacto e índice Eigenfactor.



Fonte: do autor.

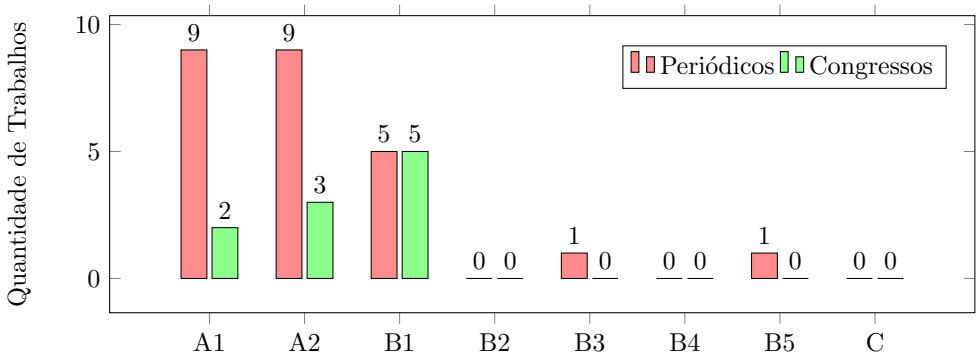
É importante destacar que o fator de impacto e o índice Eigenfactor levam em consideração apenas periódicos, não se aplicando a congressos. Assim, optou-se por considerar,

[6] Extraído de <https://jcr.clarivate.com>, acessado em 04 de Fevereiro de 2019.
[7] Extraído de <http://www.eigenfactor.org/>, acessado em 06 de Fevereiro de 2019.

também, o extrato Qualis[8] para responder a questão **QS.07**. Essa métrica é utilizada para classificar e relacionar meios de divulgação científica, portanto, é empregada na medição de periódicos e congressos. O nível de qualidade é refletido sobre os extratos A, B e C com suas próprias classificações.

O extrato Qualis é ilustrado na Figura 15. É possível observar uma quantidade relevante de publicações nos mais altos extratos de classificação. Na categoria de periódicos, alcançando o extrato A1, diversos periódicos e trabalhos foram registrados, por exemplo, *IEEE Transactions on parallel and distributed systems* com Hou, Ansari e Ren (1994), *Information Sciences* com Aguilar e Gelenbe (1997), *Computers and Operations Research* com Hwang, Gen e Katayama (2008), e *Applied Soft Computing* com Guzek et al. (2014). O extrato mais alto também foi representado na categoria de congressos com os eventos *IEEE Evolutionary Computation* (Jelodar et al. (2006)) e *IEEE Parallel and Distributed Processing* (Wang e Korfhage (1995)).

Figura 15 – Levantamento do Extrato Qualis.



Fonte: do autor.

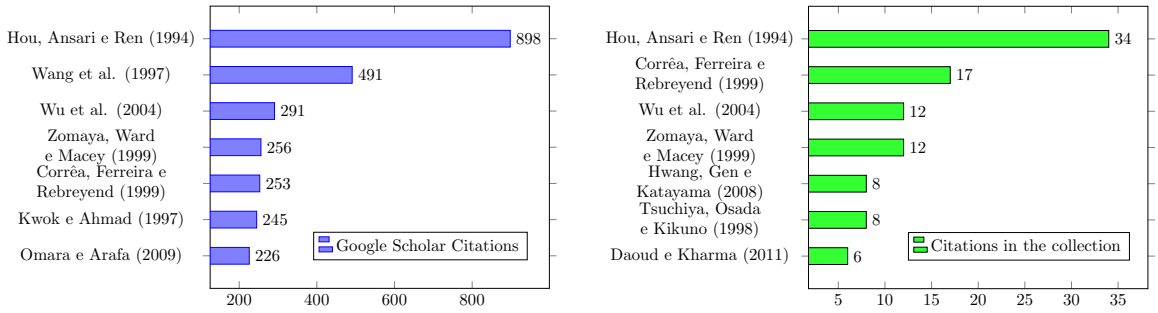
Uma publicação científica ainda pode ser avaliada ao contabilizar a sua quantidade de citações em outras publicações. Dessa forma, ao responder a **QS.08**, um levantamento foi efetuado para cada publicação da coleção. Contabilizado sobre os índices da plataforma *Google Scholar*, a Figura 16(a) relaciona os trabalhos com a maior quantidade de citações. Sendo um dos trabalhos pioneiros no tratamento do problema com o uso de GAs, o trabalho de Hou, Ansari e Ren (1994) possui a maior quantidade de citações. Também é possível observar que, com exceção dos trabalhos de Wu et al. (2004) e Omara e Arafa (2009), todos os outros trabalhos presentes no gráfico foram publicados antes do ano 2000: Hou, Ansari e Ren (1994), Wang et al. (1997), Zomaya, Ward e Macey (1999), Corrêa, Ferreira e Rebreyend (1999) e Kwok e Ahmad (1997).

Após contabilizar a quantidade geral de citações, essa revisão analisou as referências bibliográficas de cada publicação da coleção a fim de responder à **QS.09**. Logo, foi possível

[8] Extraído de: <https://sucupira.capes.gov.br>, acessado em 21 de Novembro de 2019.

contabilizar a quantidade de citações internas e classificar os trabalhos mais citados dentro da coleção. Dessa forma, o gráfico da Figura 16(b) classifica os trabalhos com a maior quantidade de citações internas. Novamente, o artigo mais referenciado é de Hou, Ansari e Ren (1994). Além de ser um dos trabalhos mais antigos da coleção, alcançou um número de 34 citações dentre os demais 63 trabalhos da coleção.

Figura 16 – Contagem da quantidade de citações.



(a) Citações no Google. (b) Citações internas.

Fonte: do autor.

Finalmente, a coleção foi utilizada para elaborar a Figura 17, que apresenta um grafo relacionando as citações internas e os trabalhos levantados. Os vértices do grafo representam os IDs dos estudos presentes na Tabela 5. É possível observar não somente os estudos mais referenciados internamente, como também os estudos que efetuam mais citações relacionadas aos demais trabalhos. Naturalmente, os estudos com a maior quantidade de citações internas recebem a maior parte das arestas de entrada.

3.2.5 Afiliações dos Pesquisadores

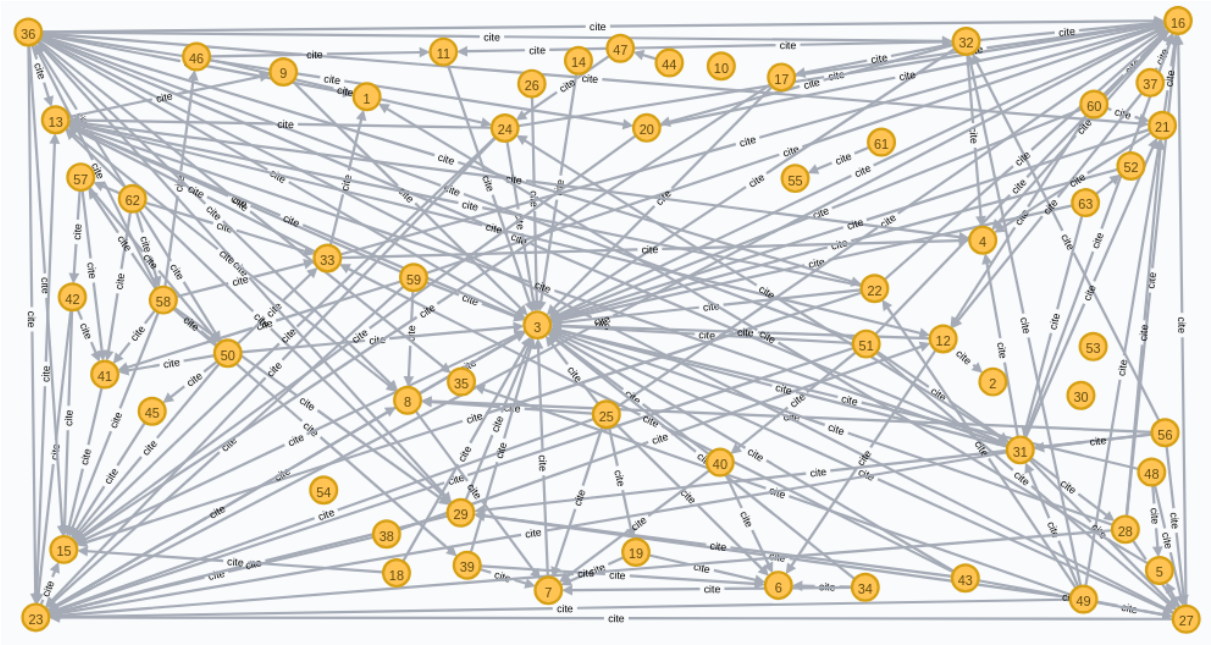
Com a formação da coleção de trabalhos, também foi possível extrair informações referentes aos autores e suas respectivas nacionalidades. A identificação da nacionalidade refere-se ao país de filiação, portanto não significa o país natal dos pesquisadores. Dessa forma, foram identificados 150 pesquisadores distintos distribuídos em 22 países. Consequentemente respondendo a **QS.10**.

A Figura 18 traz um infográfico distribuindo pesquisadores e seus respectivos países de afiliação. É possível observar que EUA e Índia são os países com a maior concentração de pesquisadores (ambos empatados com 28 autores). O esforço para melhor adaptar os GAs ao problema do MTSP pode ser demonstrado através dessa quantidade significante de pesquisadores e países envolvidos.

3.2.6 Análise de Vocabulário

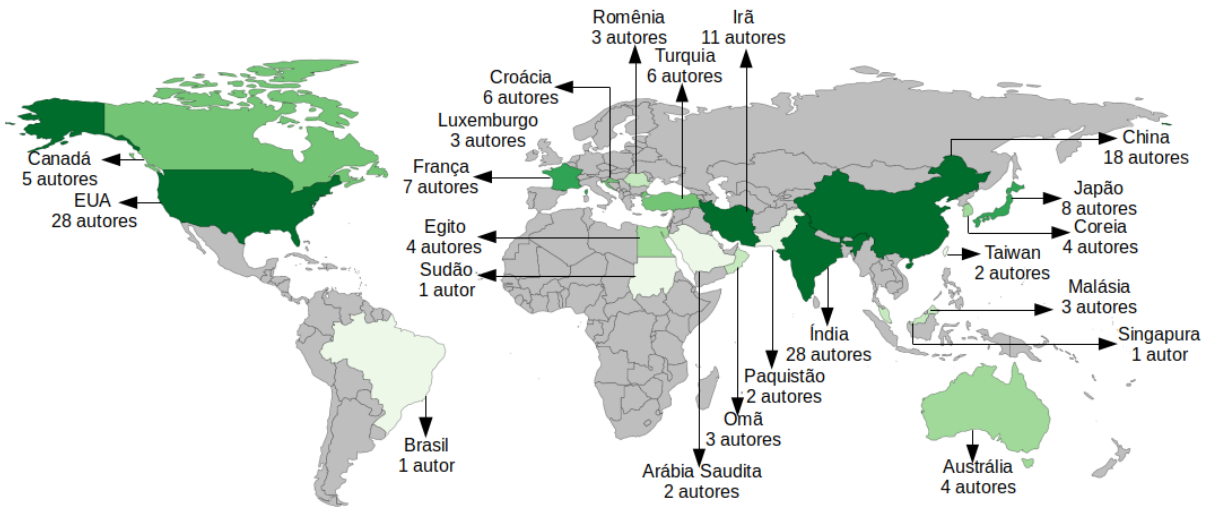
A seção de resumo de cada publicação é espaço dedicado a sintetizar as informações mais importantes de um trabalho. Além disso, geralmente, os resumos são acompanhados

Figura 17 – Grafo representando a rede de citações internas.



Fonte: do autor.

Figura 18 – Infográfico com Autores e Nacionalidades.

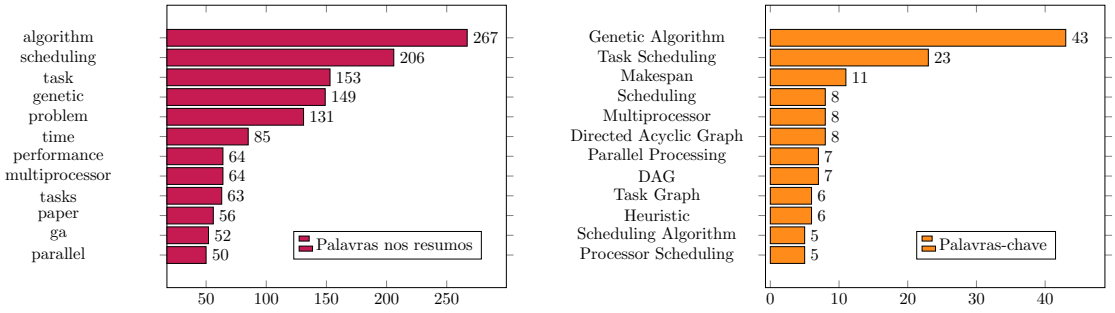


Fonte: do autor.

de um conjunto de palavras-chave que identificam aspectos da publicação. Portanto, resumos e palavras-chaves são elementos essenciais na identificação de trabalhos correlatos.

A Figura 19(a) quantifica as palavras com maior frequência encontradas nos resumos da coleção. Cabe ressaltar que os termos apresentados são exclusivamente substantivos com a exclusão de plural. As palavras mais frequentes são *algorithm*, *scheduling*, *task* e *genetic*, naturalmente associadas ao tema. Também é possível destacar outros termos relacionados ao desempenho, por exemplo *time* e *performance*. Na sequência, a Figura

Figura 19 – Quantidade de palavras nas publicações.



(a) Palavras do resumo.

(b) Palavras-chave.

Fonte: do autor.

19(b) contabiliza as palavras-chave mais frequentes na coleção. A união das quantificações responde a **QS.11**. De maneira análoga, as palavras levantadas são fortemente relacionadas ao MTSP. A identificação das principais expressões utilizadas nos resumos e nas palavras-chave ajuda no emprego de palavras frequentemente utilizadas no problema e nas bases digitais, o que pode facilitar os mecanismos de busca a encontrar e relacionar antigas e novas publicações.

3.2.7 Métodos de Seleção

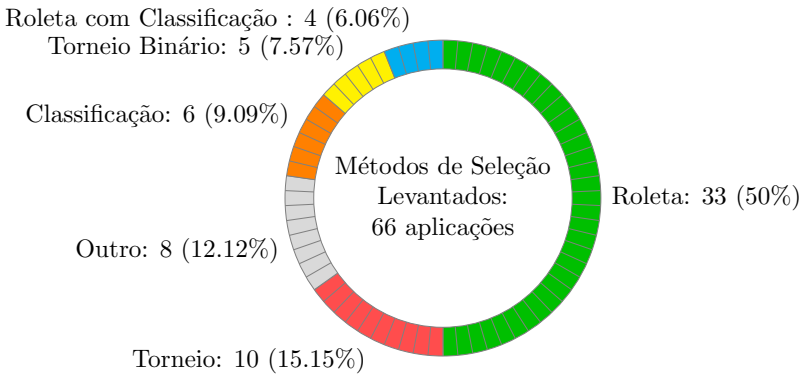
A definição dos indivíduos conduzidos aos operadores de recombinação é efetuada seguindo diversos modelos de seleção. Essas estratégias podem ser representadas como algoritmos consolidados, por exemplo os métodos de Roleta e Torneio, ou simplesmente como um mecanismo de seleção aleatória.

Complementando a resposta à **QS.01** do Capítulo 4, a Figura 20 apresenta um gráfico relacionando os principais métodos de seleção levantados na coleção. Das 66 aplicações encontradas (uma publicação pode adotar mais de uma estratégia de seleção), o algoritmo de Roleta é empregado em 33 dessas seleções ($\approx 50\%$). O segundo método mais popular levantado foi o Algoritmo de Torneio, com 10 aplicações ($\approx 15.15\%$). A fatia correspondente a outros ($\approx 12.12\%$) representa as publicações que adotam a seleção puramente aleatória em seus GAs. Por fim, também foram encontradas seleções baseadas em Classificação ($\approx 9.09\%$), combinação entre Classificação e Roleta ($\approx 6.06\%$), e Torneio Binário ($\approx 7.75\%$), onde cada torneio sempre é realizado sobre um par de indivíduos.

3.2.8 Estratégias de Elitismo

GAs podem utilizar variadas estratégias para controlar o fluxo evolutivo de uma população. Umas dessas técnicas é o elitismo, onde indivíduos com bons resultados de aptidão podem ser transportados diretamente à próxima população da evolução. Dessa forma, existem diferentes técnicas elitistas aplicadas à população, variando entre o transporte de

Figura 20 – Relação de métodos de seleção utilizados.

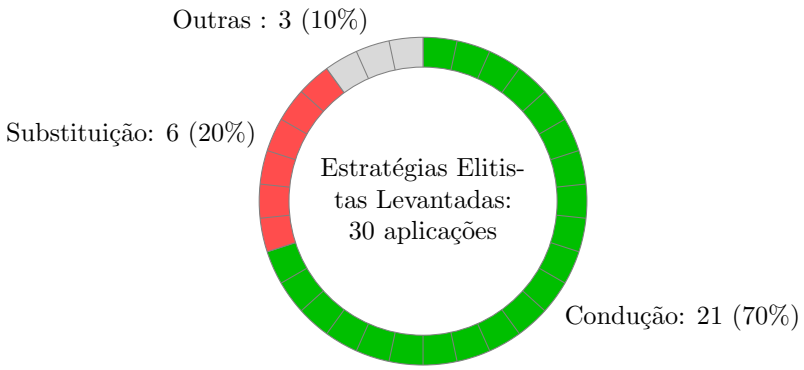


Fonte: do autor.

bons indivíduos e até mesmo a eliminação de resultados insatisfatórios. Além disso, esses mecanismos podem ser flexibilizados dependendo da estratégia ou podem ser nativos de um modelo consolidado de GA, por exemplo o elitismo empregado em algoritmos como o SPEAII e o NSGA.

O gráfico da Figura 21 relaciona as aplicações de elitismo levantadas na coleção de trabalhos, o que complementa a **QS.01**. Sendo aplicado em 21 das 30 aplicações ($\approx 70\%$), o elitismo por Condução é o modelo mais utilizado. Cabe ressaltar que essa fatia corresponde às estratégias voltadas a somente encaminhar uma quantidade fixa de indivíduos da população para a próxima geração. Em sequência, com 6 aplicações ($\approx 20\%$), o modelo de Substituição é o segundo mais aplicado. Essa estratégia não só encaminha cromossomos, como também substitui cromossomos com mal desempenho. Ao final, resta a fatia correspondente aos elitismos aplicados em algoritmos como o NSGA e o SPEAII.

Figura 21 – Relação de estratégias elitistas utilizadas.



Fonte: do autor.

Capítulo 4

Representações

Neste capítulo é apresentada uma sumarização das principais representações levantadas no processo de SLR. Uma vez que os operadores genéticos (neste caso, recombinação e mutação) estão associadas às representações, esta taxonomia se divide, em uma ordem hierárquica, entre representações e seus respectivos operadores. Este capítulo também é a resposta a primeira questão secundária de pesquisa (**QS.01**) apresentada no Capítulo 3.

A distribuição de seções deste capítulo é inspirada na divisão classificatória de representações de Guzek et al. (2014). Neste modelo, as representações são categorizadas em modelos baseados em ordenação (Seção 4.1), alocação (Seção 4.2) e na combinação de ambos (Seção 4.3). Entretanto, algumas representações levantadas na SLR possuem particularidades específicas de certos cenários. Desse modo, essas representações foram distribuídas em uma quarta categoria (Seção 4.4). Uma reunião de todas as informações levantadas no capítulo é sumarizada na Seção 4.5. Por fim, a Figura 22 apresenta dois exemplos de DAGs, sendo esses utilizados na maioria dos exemplos empregados neste capítulo.

4.1 Representações baseadas em Ordenação

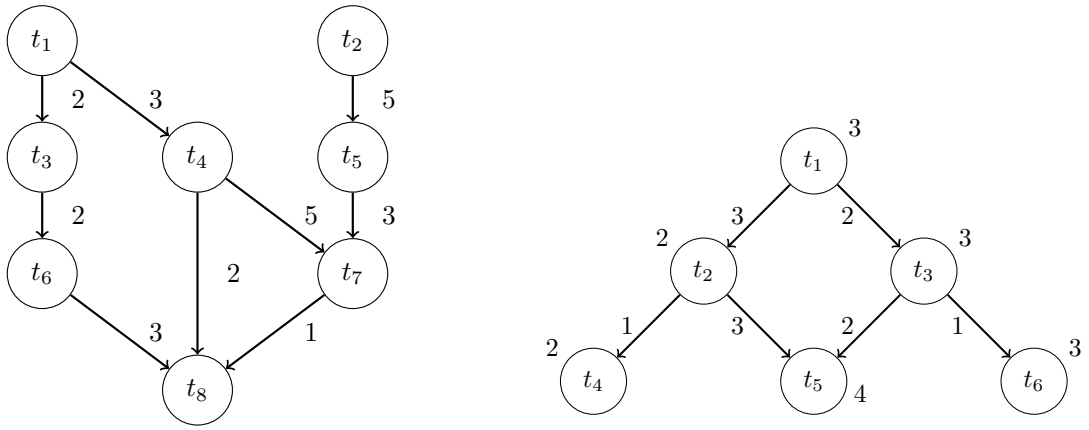
As representações apresentadas nessa seção trabalham exclusivamente com aspectos de ordenação, ou seja, a alocação de tarefas aos processadores disponíveis é realizada por um algoritmo externo. Na Subseção 4.1.1, a Codificação por Lista Ordenada é contextualizada. Em seguida, a Codificação de Prioridade é introduzida na Subseção 4.1.2.

4.1.1 Codificação por Lista Ordenada (OLE)

A Codificação por Lista Ordenada (OLE[1]) é caracterizada como uma lista completa das tarefas do grafo. A principal diferença quanto as outras codificações é a ausência

[1] Do inglês, *Ordered List Encoding*.

Figura 22 – Exemplos de grafos de tarefas.



(a) DAG com 8 tarefas. (b) DAG com 6 tarefas.

Fonte: adaptado de Hou, Ansari e Ren (1994), Tsuchiya, Osada e Kikuno (1997).

de elementos que representem a alocação dos processadores. Desta forma, a codificação OLE necessita de um método externo de alocação. A Figura 24 ilustra um exemplo da codificação OLE baseando-se no grafo da Figura 22(a). No exemplo, as tarefas dispostas no cromossomo atendem uma ordem de execução válida.

Figura 23 – Codificação de Lista Ordenada.

Índices	1	2	3	4	5	6	7	8
Tarefas	t_2	t_1	t_3	t_5	t_4	t_7	t_6	t_8

Fonte: do autor.

O trabalho de Kwok e Ahmad (1997) emprega a codificação OLE com um mecanismo de alocação denominado Minimização do Tempo de Inicialização. Nesse método, começando do primeiro nó disponível na lista de tarefas dispostas no cromossomo, cada nó é removido e, posteriormente, escalonado em algum processador que permite um menor tempo de inicialização (KWOK; AHMAD, 1997). O Algoritmo 3 sumariza as instruções da Minimização do Tempo de Inicialização, onde DAT[2] identifica o tempo em que os dados de um tarefa t_i ficam disponíveis em um determinado processador escalonado pe; $w(t_i)$ obtém o custo computacional de uma tarefa t_i; e ST indica o tempo de finalização mínimo (min_ST) e o atual calculado ($atual\_ST$).

Ramachandra e Elmaghraby (2006) empregam uma variação de codificação OLE, onde as alocações de processadores são definidas através do algoritmo da regra FAM[3], onde

[2] Do inglês, *Data Available time*.
[3] Do inglês, *First Available Machine*.

Algoritmo 3 Minimização do Tempo de Inicialização – adaptado de Kwok e Ahmad (1997)

entrada: Lista de escalonamento L;
saída: pe_s escalonados;

```
1: ∀j tempoPronto(pe_j) = 0;
2: enquanto a lista de escalonamento L não for vazia faça
3:    remova o primeiro nó t_i de L;
4:    Min_ST = ∞;
5:    para j = 0 até p − 1 faça
6:       atual_ST = max{tempoPronto(pe_j), DAT(t_i, pe)};
7:       se atual_ST < Min_ST então
8:          Min_ST = atual_ST;
9:          candidato = pe_j;
10:      fim se
11:   fim para
12:   escalonar t_i para candidato;
13:   tempoPronto(candidato) = Min_ST + w(t_i);
14: fim enquanto
15: retorne: pe_s escalonados;
```

tarefas são distribuídas às primeiras máquinas disponíveis. Outra diferença é o controle das restrições de precedência, sendo nesta versão implementado através de uma matriz de adjacência MA. Essa matriz é formada pela relação tarefa-tarefa e contém zeros e uns, onde o valor 1 em uma célula (i, j) representa que a tarefa t_i precede a tarefa t_j, em outras palavras, $i \prec j$. O trabalho de Ramachandra e Elmaghraby (2006) ainda emprega um controle sobre os possíveis intervalos em que uma tarefa pode ser deslocada sem causar inconsistências nas restrições de precedência. Essas informações são passadas ao operador de mutação. Para tal, são definidos os limites EP e LP[4], respectivamente, a primeira e a última posição do intervalo de deslocamento. A Figura 24 apresenta exemplos da matriz de adjacência e dos limites EP e LP aplicados ao grafo da Figura 22(a).

Awadall, Ahmad e Al-Busaidi (2013) empregam a OLE com uma heurística min-min como mecanismo de ordenação. Segundo a definição de Awadall, Ahmad e Al-Busaidi (2013), criada para mapear tarefas em sistemas heterogêneos, a heurística min-min encontra o tempo de conclusão mínimo de todas as tarefas não mapeadas e, posteriormente, a tarefa com o menor tempo de conclusão é selecionada e mapeada para um processador do conjunto. Ao final, a tarefa é removida e o processo é executado novamente até que todas as tarefas sejam mapeadas. Nessa heurística, o tempo de conclusão é igual ao tempo de execução da tarefa na máquina em adição ao tempo de conclusão de todas as outras tarefas mapeadas para a mesma máquina.

Xu et al. (2012) e Xu et al. (2014) apresentam outra codificação análoga, onde um cromossomo é designado como uma fila de prioridades. Em Xu et al. (2012), com a

[4] Do inglês, *Earliest Position* e *Latest Position*.

Figura 24 – Controles adicionais na OLE de Ramachandra e Elmaghraby (2006).

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
t_1	0	1	1	1	0	1	1	1
t_2	0	0	0	0	1	0	1	1
t_3	0	0	0	0	0	1	0	1
t_4	0	0	0	0	0	0	1	1
t_5	0	0	0	0	0	0	1	1
t_6	0	0	0	0	0	0	0	1
t_7	0	0	0	0	0	0	0	1
t_8	0	0	0	0	0	0	0	0

(a) Matriz de adjacência.

Tarefas	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
EP	1	1	2	2	2	3	5	8
LP	3	5	6	6	6	7	7	8

(b) Intervalos de deslocamento.

Fonte: do autor.

geração da população inicial baseada em uma heurística de classificação ascendente, os demais cromossomos, criados através de recombinação e mutação, perpetuam escalonamentos válidos. De forma semelhante, Xu et al. (2014) adota heurísticas de classificação ascendente, classificação descendente e uma combinação entre nível e classificação ascendente-descendente. Outro diferencial nesta codificação é o emprego da heurística HEFT (TOPCUOGLU; SEVILMIS, 2002) para computação da aptidão.

O trabalho de Hassan et al. (2015) apresenta variados GAs com diferentes codificações, dentre esses, o primeiro GA apresentado implementa a codificação OLE. Em contrapartida, após a execução das operações de reprodução, cromossomos inválidos podem ser perpetuados na população. Para filtrar cromossomos válidos, Hassan et al. (2015) aperfeiçoa o GA para atribuir um grande valor de makespan aos cromossomos inválidos identificados na etapa de mutação.

Finalmente, Mohamed e Awadalla (2011) utilizam a codificação de listas topológicas dentre as três codificações utilizadas no seus respectivos trabalhos.

4.1.1.1 Recombinação de Ponto Único com Ordenação (SPX)

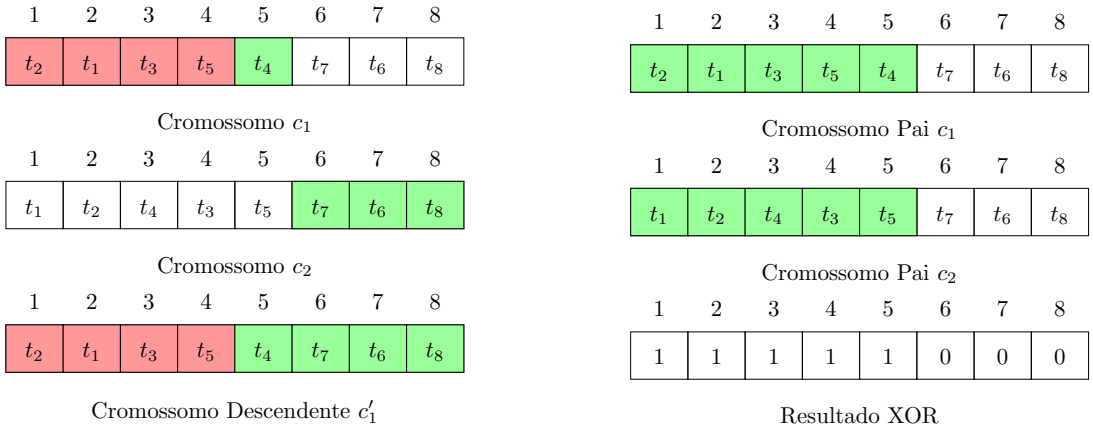
Kwok e Ahmad (1997) observam a aplicação de três operadores de recombinação: PMX, CX e SPX com ordenação. Os autores demonstram a inviabilidade da aplicação dos dois primeiros, uma vez que os cromossomos resultantes dessas operações tornam-se soluções infactíveis. Portanto, o SPX com ordenação é adotado devido a geração de novos cromossomos válidos.

Nessa recombinação, um único ponto de corte é definido aleatoriamente. Em seguida, um par de cromossomos, também selecionado aleatoriamente, é dividido em duas partes, esquerda e direita. A divisão é realizada sob o ponto de corte definido no início da operação. O cromossomo descendente é formado pela junção da parte à esquerda do primeiro

cromossomo pai e da parte à direita do segundo cromossomo pai. Genes duplicados no segmento à direita do cromossomo descendente são sobrescritos observando os genes ausentes que estejam no segundo pai. Esse processo garante a criação de soluções válidas com completude, singularidade e exatidão. Para a geração de um segundo cromossomo descendente, o operador é aplicado ao inverter o par de cromossomos pais.

Um exemplo da recombinação de um ponto baseada em ordem é ilustrado, sobre dois cromossomos válidos, na Figura 25(a). O primeiro segmento à esquerda do cromossomo pai c_1, t_2, t_1, t_3 e t_5, é copiado inteiramente para o cromossomo descendente. Logo após, o segundo segmento à direita do cromossomo pai c_2, t_5, t_7, t_6 e t_8, também é copiado ao cromossomo descendente. Ao final, o gene duplicado à direita do descendente c'_1, t_5, é sobrescrito observando-se o mesmo gene no cromossomo pai c_1 e qual gene está disposto no cromossomo pai c_2 no mesmo índice, isto é, a tarefa t_3. Entretanto, t_3 também está duplicada, de forma que a correção é reexecutada, considerando t_3, e ao final, t_4 é escrita no cromossomo descendente.

Figura 25 – Recombinação de Ponto Único com Ordenação.



(a) Exemplo da recombinação. (b) Excluindo clonagem.

Fonte: adaptado de Kwok e Ahmad (1997).

Ramachandra e Elmaghraby (2006), Xu et al. (2012), Xu et al. (2014) , Mohamed e Awadalla (2011), Awadall, Ahmad e Al-Busaidi (2013) e Hassan et al. (2015) são outros exemplos de trabalhos que implementam o SPX com ordenação. Em adição, Xu et al. (2014) efetua um pré-processamento para evitar a clonagem de cromossomos. O procedimento é uma operação XOR ("ou exclusivo") entre um par de cromossomos selecionados pra recombinação. Dessa forma, a recombinação só implementará pontos de cortes dispostos na coleção de 1s retornada pela operação XOR. A Figura 25(b) ilustra um exemplo da exclusão de pontos de clonagem considerando os cromossomos válidos da Figura 25(a). Por fim, Xu et al. (2014) também sumariza as instruções executadas na da recombinação de um ponto conforme apresentado no Algoritmo 4.

Algoritmo 4 Recombinação de um ponto baseada em Ordem – adaptado de Xu et al. (2014)

entrada: Cromossomos c_1 e c_2;
saída: Cromossomos descendentes c'_1 e c'_2;
1: Selecione aleatoriamente um ponto de corte x adequado;
2: Divida o cromossomo c_1 e o cromossomo c_2 nos segmentos esquerdo e direito;
3: Gere um novo cromossomo descendente c'_1;
4: Copie o segmento esquerdo do cromossomo c_1 ao segmento esquerdo de c'_1;
5: Copie os genes do cromossomo c_2, que não estejam no segmento à esquerda do cromossomo c_1, para o segmento à direita no cromossomo c'_1;
6: Gere um novo cromossomo descendente c'_2;
7: Copie o segmento esquerdo do cromossomo c_2 ao segmento esquerdo de c'_2;
8: Copie os genes do cromossomo c_1, que não estejam no segmento à esquerda do cromossomo c_2, para o segmento à direita do cromossomo c'_2;
9: **retorne:** c'_1 e c'_2;

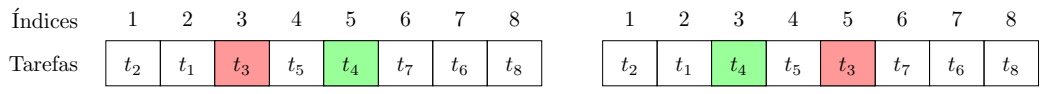
4.1.1.2 Mutações

As próximas subseções apresentam diversas mutações empregadas na OLE. Inicialmente, são introduzidas mutações baseadas no deslocamento de uma tarefa dentro da cadeia do cromossomo. Ao final, uma mutação baseada na troca de tarefas é contextualizada.

Mutação de Deslocamento de Tarefas (STM)

A STM proposta no trabalho de Kwok e Ahmad (1997) é aplicada em cromossomos válidos e não permite a geração de soluções inválidas. O operador atua como uma permutação entre um par aleatório de tarefas dispostas no cromossomo. A limitação do operador é atribuída às tarefas que não podem ser permutadas. Tarefas do grafo são permutáveis se não pertencem ao mesmo caminho no grafo de tarefas (KWOK; AHMAD, 1997). Como mecanismo de verificação de precedência, Kwok e Ahmad (1997) executam uma busca em profundidade no grafo de tarefas. A Figura 26 apresenta um exemplo do operador onde as tarefas permutáveis t_3 e t_4 (Figura 22(a)) são selecionadas e trocadas aleatoriamente.

Figura 26 – Mutação de Deslocamento de tarefas na OLE.



Fonte: adaptado de Kwok e Ahmad (1997).

O trabalho de Ramachandra e Elmaghraby (2006) também implementa o mesmo modelo de mutação. Entretanto, eles empregam o conceito de fechamento transitivo do grafo

de tarefas para determinar, em tempo constante, se duas tarefas selecionadas aleatoriamente são permutáveis.

Mutação de Deslocamento de Tarefas com Sucessão (STM)

A STM implementada no trabalho de Xu et al. (2012) define uma outra forma de realizar as permutações. Neste operador, dado um cromossomo da população, uma tarefa (gene) t_i é selecionada aleatoriamente. Em seguida, o algoritmo busca o primeiro sucessor de t_i entre a própria tarefa t_i até o extremo final do cromossomo. Posteriormente, uma tarefa $t_j \in (t_i + 1, Succ(t_i) - 1)$ é aleatoriamente selecionada, e sua tarefa predecessora $Pred(t_j)$ é buscada entre $Succ(i)$ e o extremo começo do cromossomo. Por fim, se a $Pred(t_j) < t_i$, as tarefas t_i e t_j podem ser permutadas. O Algoritmo 5 enumera as instruções da mutação empregada por Xu et al. (2012).

Algoritmo 5 Mutação por Deslocamento de Tarefas com Sucessão – adaptado de Xu et al. (2012) e Xu et al. (2014).

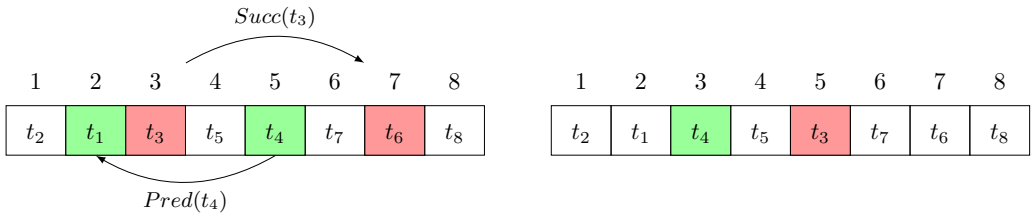
entrada: Cromossomo selecionado aleatoriamente;
saída: Cromossomo descendente;
1: Selecione aleatoriamente um gene i;
2: Encontre o primeiro sucessor $Succ(i)$ entre i até o final;
3: Selecione aleatoriamente um gene $j \in (i + 1, Succ(i) - 1)$;
4: Encontre o primeiro predecessor $Pred(j)$ entre $Succ(i)$ até o começo;
5: **se** $Pred(j) < i$ **então**
6: Troque as posições entre o gene i e o gene j;
7: Gera um novo Cromossomo descendente;
8: **fim se**
9: **retorne:** Cromossomo descendente;

A Figura 27 ilustra um exemplo do operador sobre o grafo da Figura 22. Inicialmente, o gene aleatoriamente selecionado pertence a tarefa t_3. Logo após, $Succ(t_3)$ é encontrado ao percorrer o grafo. Então, a tarefa t_4 é selecionada entre o intervalo $t_3 + 1$ e $t_6 - 1$. A tarefa $Pred(t_4)$ é encontrado logo em seguida. Ao final, conforme o Algoritmo 5, $Pred(t_4)$ é menor que t_3, portanto, t_3 e t_4 são permutadas, configurando o cromossomo à direita na Figura 27.

Mutação de Troca (SM)

Os trabalhos de Mohamed e Awadalla (2011) e Awadall, Ahmad e Al-Busaidi (2013) comparam variadas codificações, sendo essas PLE, OLE e MSE. Nos modelos OLE e MSE, a Mutação por Troca (SM) é empregada, onde um par de genes é, aleatoriamente, selecionado. Posteriormente, as tarefas referentes a esses genes são trocadas. Portanto, a SM afeta mais de um gene na sua aplicação, diferentemente da STM. Além disso, as

Figura 27 – Mutação de Deslocamento de tarefas com Sucessão na OLE.



Fonte: adaptado de Xu et al. (2012).

mudanças devem obedecer as restrições de precedência, seja por limitações na aplicação do operador ou pelo uso de mecanismos reparadores.

4.1.2 Codificação de Prioridade (PE)

Conforme a implementação de Hwang, Gen e Katayama (2008), a Codificação de Prioridade (PE[5]) define índices para as tarefas que possam identificar níveis de antecedência. Dispostos em uma lista de tarefas, essas valores de prioridade são gerados aleatoriamente e dependem da quantidade de processadores.

Inicialmente, após a definição dos valores de prioridade, permutações podem ser executadas. O Algoritmo 6 enumera as instruções necessárias para a geração e permutação desses valores. É possível observar nas instruções que os valores de prioridade são singulares, uma vez que são uma combinação da quantidade de processadores, identificadores das tarefas e algum valor aleatório limitado. Além disso, a segunda parte do Algoritmo 6 demonstra que as permutações são executadas aleatoriamente.

Algoritmo 6 Definição da Codificação de Prioridade – adaptado de Hwang, Gen e Katayama (2008)

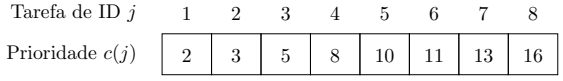
entrada: número de processadores m, número de tarefas n;
saída: cromossomo $c(j)$

para $j = 1$ **até** n **faça**
  $c(j) \leftarrow m \times j - \text{sortear}[0, m-1]$;
fim para
para $i = 1$ **até** $[n/2]$ **faça**
  $j \leftarrow \text{sortear}[1, n]$;
  $k \leftarrow \text{sortear}[1, n]$;
  se $j \neq k$ **então**
    $\text{trocar}(c(j), c_k(k))$;
  fim se
fim para
retorne: cromossomo $c(j)$;

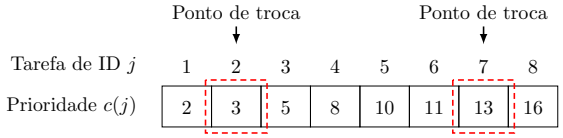
[5] Do inglês, *Priority Encoding*.

A Figura 28 apresenta um exemplo que divide as fases da aplicação do Algoritmo 6. Em resumo, a Figura 28(a) apresenta um cromossomo com valores de prioridades gerados considerando um ambiente com dois processsadores ($m = 2$). Em seguida, a Figura 28(b) demonstra uma permutação de prioridade executada sobre as tarefas t_2 e t_7. Ao final, o cromossomo resultante é ilustrado na Figura 28(c), configurando também, o final da execução do Algoritmo 6.

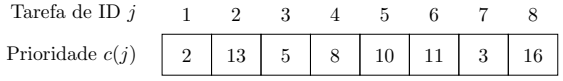
Figura 28 – Codificação de Prioridade.



(a) Primeira fase do algoritmo 6.



(b) Segunda fase do algoritmo 6.



(c) Cromossomo após permutação.

Fonte: adaptado de Hwang, Gen e Katayama (2008).

Após a fase de geração, um cromossomo pode sofrer a aplicação dos operadores de recombinação e mutação. Em contrapartida, alguns procedimentos são necessários para decodificar o cromossomo em um escalonamento válido. Nesse caso, inicialmente, é necessário gerar uma lista com uma sequência de execução que englobe todas tarefas do grafo. Essa sequência é influenciada pelos valores de prioridade contidos no cromossomo. As instruções para a geração da sequência de execução são enumeradas no Algoritmo 7.

O próximo passo da decodificação configura a alocação das tarefas em seus respectivos processadores. Nesta fase, conforme destacado por Hwang, Gen e Katayama (2008), é necessário checar as relações de precedência e adicionar o custo de comunicação. Além de alocar as tarefas, este procedimento garante o cálculo do custo de execução e a montagem do escalonamento final. O Algoritmo 8 sumariza as instruções desse último passo da decodificação.

4.1.2.1 Recombinação Mapeada Ponderada (WMX)

Disposta nos trabalhos de Hwang, Gen e Katayama (2008) e Azghadi et al. (2008), a principal recombinação encontrada na PE, denominada como Recombinação Mapeada

Algoritmo 7 Decodificação da lista – adaptado de Hwang, Gen e Katayama (2008)

entrada: número de tarefas n, cromossomo $c(j)$, o conjunto de nós S';
saída: sequência de tarefas TS

```
1: S' ← ∅, TS ← ∅;
2: n ← 0, j ← 0;
3: enquanto j ≤ n faça
4:     S' ← S' ∪ suc_j;
5:     j* ← arg max(v(j)|j ∈ S);
6:     S' ← S'\j*;
7:     TS ← TS ∪ j*;
8:     j ← j*;
9: fim enquanto
10: retorne: sequência de tarefas TS
```

Ponderada (WMX), utiliza mapeamentos para realizar permutações nos pares de cromossomos selecionados. O operador pode ser visto como uma operação de remapeamento por referência e como um operador de recombinação de dois pontos aplicado em cadeias binárias (HWANG; GEN; KATAYAMA, 2008). A sequência de instruções da recombinação baseada em mapeamento de pesos é enumerada no Algoritmo 9.

Um exemplo da execução do algoritmo 9 é demonstrado na Figura 29. Inicialmente, na Figura 29(a), os cromossomo c_1 e c_2 são selecionados para o cruzamento. Com a definição dos pontos de corte, um intervalo separa as cadeias que terão seu valores permutados. Em seguida, a permutação é realizada considerando os valores de prioridade definidos em cada gene. No exemplo da Figura 29(b), os genes do cromossomo c_1, 5, 8, 10 e 11, refletem como ordem de prioridade a sequência 4, 3, 2 e 1, respectivamente. Essa ordem de prioridade é utilizada na permutação dos genes dispostos no cromossomo c_2. De modo igual, a ordem de prioridade dos cromossomos de c_2 (3, 4, 1 e 2) reflete a permutação de cromossomo em c_1. Ao final do processo, Figura 29(c), os descendentes c'_1 e c'_2 são formados.

4.1.2.2 Mutações

Denominado Mutação de Prioridade de Tarefas (PTM[6]), o método de mutação empregado na PE é uma variação da troca de tarefas. Nesta implementação, as prioridades são trocadas aleatoriamente. Essa permutação pode influenciar na ordem de tarefas, Algoritmo 7, e na alocação de processadores. O Algoritmo 10 apresenta as instruções para a mutação baseada em troca implementada por Hwang, Gen e Katayama (2008). Outro operador implementado é a Mutação de Embaralhamento de Prioridade (SPM[7]), sugerido por Azghadi et al. (2008), o método seleciona aleatoriamente uma região do cromossomo onde as prioridades serão permutadas.

[6] Do inglês, *Priority Task Mutation.*
[7] Do inglês, *Shuffle Priority Mutation.*

Algoritmo 8 Montagem do escalonamento – adaptado de Hwang, Gen e Katayama (2008)

entrada: Tempo de processamento p_k, sequência de tarefas TS, cromossomo $c(j)$, custo de comunicação CC_{j_k};
saída: makespan f, escalonamento S;

```
1: P_i ← 0, i = 1, 2, ..., m;;
2: t_k ← 0, k = 1, 2, ..., n;
3: s ← 0;
4: para j = 1 até n faça
5:    s ← TS_i;
6:    P_i ← c(s) mod m;
7:    se P_i = 0 então
8:       P_i ← m;
9:    fim se
10:   se tarefa alocada T_j ≺ T_k então
11:      se P_i ≠ P_l então
12:         e_k ← max{t_j | j ∈ pre(k)} + CC_{j_k};
13:         t_k ← e_k + p_k;
14:         S ← S ∪ S_k{P_i; e_k + p_k};
15:      senão
16:         e_k ← max{t_j | j ∈ pre(k)};
17:         t_k ← e_k + p_k;
18:         S ← S ∪ S_k{P_i; e_k + p_k};
19:      fim se
20:   fim se
21: fim para
22: retorne: escalonamento S
23: makespan ← max(t_k, k = 1, 2, ..., n);
```

4.2 Representações baseadas em Alocação

De forma análoga à seção anterior, esta seção apresenta representações que codificam exclusivamente aspectos de alocação. Portanto, conforme representações baseadas em ordenação, as representações baseadas em alocação também necessitam de um algoritmo externo, porém, voltado à ordenação das tarefas. Inicialmente, a Subseção 4.2.1 contextualiza a Codificação por Lista de Processadores. Em seguida, a Subseção 4.2.2 introduz a Codificação de Tarefa Fixa com Alocação. Por fim, na Subseção 4.2.3, a Codificação de Intervalos é apresentada.

4.2.1 Codificação por Lista de Processadores (PLE)

Dentre as representações utilizadas para resolver o MTSP, existem codificações baseadas somente no processo de alocação. Neste caso, a ordenação topológica pode ser estabelecida por alguma heurística de ordenação, ao passo que a alocação de processadores às tarefas seja definida na codificação do cromossomo Alguns exemplos de trabalhos

Algoritmo 9 Recombinação por mapeamento de peso – adaptado de Hwang, Gen e Katayama (2008)

entrada: cromossomos pais c_1 e c_2, número de tarefas n;
saída: descendentes c'_1 e c'_2;

$s \leftarrow \text{sortear}[1, n-1]$;
$t \leftarrow \text{sortear}[s+1, n]$;
$l \leftarrow t - s$;
para $i = 1$ **até** l **faça**
    $s_1[i] \leftarrow c_1[s+i]$;
    $s_2[i] \leftarrow c_2[s+i]$;
fim para
$s_1[.] \leftarrow \text{ordenar}(s_1[.])$;
$s_2[.] \leftarrow \text{ordenar}(s_2[.])$;
$c'_1 \leftarrow c_1[1:s-1] \quad // \quad c_2[s:t] // c_1[t+1:n]$;
$c'_2 \leftarrow c_2[1:s-1] \quad // \quad c_1[s:t] // c_2[t+1:n]$;
para $i = 1$ **até** l **faça**
    para $j = 1$ **até** l **faça**
        se $c'_1[s+i] = s_2[j]$ **então**
            $c'_1[s+i] \leftarrow s_1[j]$;
        fim se
    fim para
    para $j = 1$ **até** l **faça**
        se $c'_2[s+i] = s_1[j]$ **então**
            $c'_2[s+i] \leftarrow s_2[j]$;
        fim se
    fim para
fim para
retorne: descendentes c'_1 e c'_2

Algoritmo 10 Mutação de prioridade– adaptado de Hwang, Gen e Katayama (2008)

entrada: cromossomo c, l
saída: descendente c'

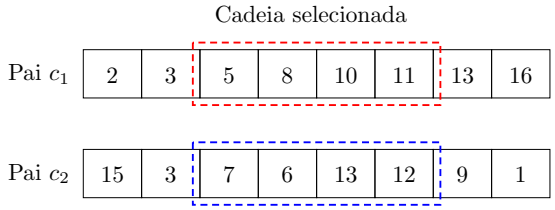
$i \leftarrow \text{sortear}[1 : l-1]$;
$j \leftarrow \text{sortear}[i+1 : l]$;
$c \leftarrow c[1:i-1] // c[j] // c[i+1:j-1] // c[i] // c[j+1:l]$;
retorne: descendente c';

que utilizam essa representação são Benten e Sait (1994), Aguilar e Gelenbe (1997), Wang et al. (2011) e Hassan et al. (2015)

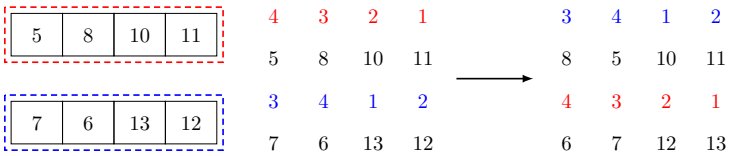
Na Codificação por Lista de Processadores (PLE[8]), cada locus carateriza as n tarefas dispostas no DAG. Dessa forma, ao contemplar todas as tarefas do grafo e não habilitar a duplicação de tarefas, o tamanho final do cromossomo será n. Os alelos do cromossomo podem assumir valores entre 1 e k, sendo k o total de processadores adotados ao cenário. Em aspecto computacional, o cromossomo pode ser codificado como uma coleção de

[8] Do inglês, *Processor List Encoding*.

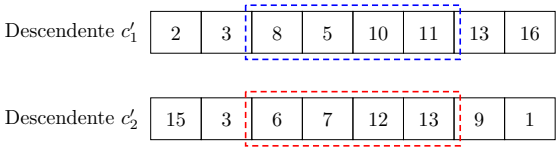
Figura 29 – Recombinação Mapeada Ponderada.



(a) Definindo intervalos do corte.



(b) Efetuando a troca.



(c) Descendentes gerados.

Fonte: adaptado de Hwang, Gen e Katayama (2008).

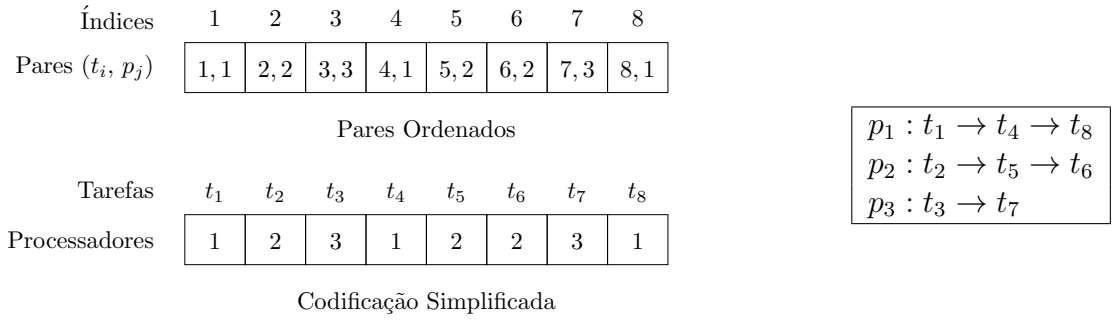
números inteiros com a possibilidade de repetição.

A Figura 30(a) contempla um exemplo da PLE. O exemplo é construído sob o grafo da Figura 22(a) e apresenta uma simplificação observada em Benten e Sait (1994): após a atribuição dos processadores às tarefas disponíveis, será possível obter uma lista de pares que combina tarefa e processador. Por exemplo, a lista de alocações: (t_1,1), (t_2,2), (t_3,3), (t_4,1), (t_5,2), (t_6,2), (t_7,3), (t_8,1). Uma vez que essa lista ordene suas tarefa por ordem de precedência, será possível simplificar a codificação, conforme ilustrado no limite inferior da Figura 30(a).

De forma similar, Wang et al. (2011) empregam a PLE ao simplificar uma relação bidimensional entre alocação e ordem, conforme demonstrado na Figura 30(b), onde a relação bidimensional é convertida à codificação simplificada presente na Figura 30(a).

O terceiro e quarto algoritmos apresentados por Hassan et al. (2015) empregam a codificação PL. Em ambas as técnicas, soluções válidas são produzidas baseando-se em heurísticas adicionais. O terceiro algoritmo utiliza a densidade local do grafo para determinar o escalonamento, enquanto que o quarto algoritmo combina a densidade local com uma heurística de tempo mais ágil de conclusão (ECT). A PLE também é implementada dentre as três codificações utilizadas por Mohamed e Awadalla (2011) e Awadall, Ahmad e Al-Busaidi (2013).

Figura 30 – Construção da Codificação de Lista de Processadores.



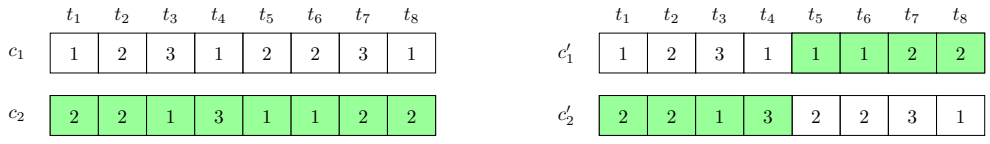
(a) Simplificando a codificação (b) Relação de precedência.

Fonte: do autor.

4.2.1.1 Recombinação de Ponto Único (SPX)

O operador SPX é o modelo adotado em Benten e Sait (1994), Wang et al. (2011), Mohamed e Awadalla (2011), Awadall, Ahmad e Al-Busaidi (2013) e Hassan et al. (2015). Hassan et al. (2015) também adota esse método, no entanto, apenas nas codificações do terceiro e quarto GA apresentados. Conforme o método empregado nas outras codificações, um ponto de corte é definido aleatoriamente. Em seguida, o ponto de corte divide dois cromossomos pais em duas metades, esquerda e direita. Ao final, o operador cria dois cromossomos descendentes, onde ambos são formados por um segmento de um dos pais em combinação ao segmento do outro pai. Benten e Sait (1994) acrescentam que a recombinação de um ponto de corte é geneticamente efetiva, computacionalmente barata e sempre produz escalonamentos válidos. A Figura 31 sintetiza um exemplo da recombinação de um ponto em dois cromossomos, c_1 e c_2.

Figura 31 – Recombinação de Ponto Único na PLE.



Fonte: do autor.

4.2.1.2 Mutações

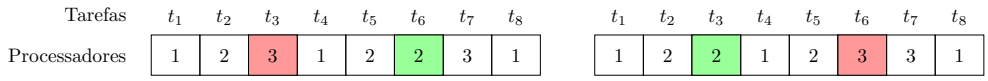
Utilizando o modelo SPX para recombinações, PLE possui três modelos de mutação apresentados nas próximas subseções. Primeiramente, uma mutação baseada na troca de processadores alocados é apresentada. Em seguida, uma mutação baseada na realocação

aleatória de processadores é contextualizada. Por fim, a última subseção apresenta uma mutação de seleção de realocações baseando-se em prioridade.

Mutação de Troca de Processadores (SPM)

O primeiro mecanismo de mutação adotado nessa codificação configura uma simples troca de processadores alocados dentro de um cromossomo, conforme a SPM. Conforme a descrição de Benten e Sait (1994), se uma tarefa t_i estiver alocada em um processador p_l, e uma tarefa t_j estiver alocada a um processador p_k, após a aplicação do operador, t_i será alocada ao processador p_k, enquanto que t_j será designada o processador p_l. A Figura 32 ilustra um exemplo, onde as tarefas t_3 e t_6 têm seus processadores permutados. Mohamed e Awadalla (2011) e Awadall, Ahmad e Al-Busaidi (2013) também adotam este operador ao implementarem a PLE.

Figura 32 – Mutação de Troca de processadores na TLE.

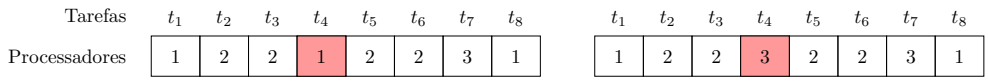


Fonte: do autor.

Mutação de Processador (PM)

Hassan et al. (2015) emprega a atribuição de novos processadores no terceiro e quarto algoritmos apresentados. Nesse modelo de mutação (PM), um gene é aleatoriamente selecionado para, posteriormente, ter um novo processador definido aleatoriamente. A Figura 33 ilustra um exemplo de aplicação do operador, de forma que um novo processador aleatório é definido para a tarefa t_4. Mohamed e Awadalla (2011), Awadall, Ahmad e Al-Busaidi (2013) e Hassan et al. (2015) são outros exemplos de trabalhos que implementam este operador ao empregarem a codificação de lista de processadores.

Figura 33 – Mutação por sorteio de processador na codificação de lista de processadores.



Fonte: adaptado de Hassan et al. (2015).

Mutação de Alocação com Prioridade (PAM)

A Mutação de Alocação com Prioridade (PAM[9]) é apresentada no trabalho de Wang et al. (2011), em que a otimização é realizada nas métricas de makespan e confiabilidade. Com base nessa otimização, Wang et al. (2011) propõe uma forma de mutação orientada a selecionar o processador mais viável, isto é, neste contexto, o processador com o valor mínimo de multiplicação entre velocidade de instrução, γ, e taxa de falha, rdr, deve ser selecionado. O método é implementado sobre uma heurística de prioridade, onde $1/(\gamma_i \cdot rdr_i)$ define a prioridade de um processador (recurso) r_i. Wang et al. (2011) destaca que, se um escalonamento s alocar todas as tarefas para processadores com a mais alta prioridade, um outro escalonamento $s' \neq s$ com confiabilidade $R_{s'}$ será tal como $R_{s'} < R_s$. Em resumo, o operador seleciona aleatoriamente uma tarefa e, em seguida, realoca a tarefa selecionada para um processador com o menor valor de $\gamma_i \cdot rdr_i$.

4.2.2 Codificação de Tarefa Fixa com Alocação (FTE)

O trabalho de Jelodar et al. (2006) elabora uma forma de codificação que utiliza TD, denominada Codificação de Tarefa Fixa com Alocação (FTE[10]). Essa codificação estabelece um espaço fixo para as tarefas ao mesmo tempo em que define, estaticamente, as duplicações de tarefas, utilizando informações dispostas no DAG. Seguindo a definição de Jelodar et al. (2006), na FTE, um cromossomo é dividido em duas partes. A primeira parte se refere à alocação de processadores nas tarefas; a segunda parte lida com o tratamento da TD. Um exemplo da codificação FTE, baseado no grafo de tarefas da Figura 22(a), é ilustrado na Figura 34.

Figura 34 – Codificação de Tarefa Fixa com Alocação.

1	2	3	4	5	6	7	8	1.1	4.2
2	1	2	3	4	2	1	4	3	2

Fonte: adaptado de Jelodar et al. (2006).

Considerando as tarefas e suas respectivas restrições de precedência, a primeira parte do cromossomo na Figura 34 distribui as tarefas e as identifica através do índice da coleção. Paralelamente, cada célula da coleção determina o processador onde a tarefa será alocada. Desta forma, ao assumir um cenário com 4 processadores, o conteúdo das células varia de 1 a 4. Conforme destacado por Jelodar et al. (2006), essa primeira seção do cromossomo mantêm um conjunto com todas as tarefas abordadas enquanto que não perde a liberdade para buscar por mais soluções otimizadas.

[9] Do inglês, *Priority Allocation Mutation*.
[10] Do inglês, *Fixed Task Encoding*.

A segunda parte do cromossomo lida com a TD e pode ter comprimento variado. Jelodar et al. (2006) adicionam uma restrição à TD: uma tarefa só pode ser duplicada se e somente se o custo de comunicação entre os processadores for maior que o custo de execução da tarefa. Além disso, baseando-se nas restrições de precedência, uma tarefa não precisa ser duplicada nas seguintes situações:

- **Folha no grafo:** tarefas que não possuem tarefas descendentes, isto é, uma tarefa que não é pré-requisito para nenhum outra tarefa;
- **Filha única:** tarefas que possuem somente uma única tarefa descendente. A tarefa pode ser alocada em um processador juntamente com sua tarefa descendente.

Ao contrário das regras supracitas, se uma tarefa for requisitada por mais de uma tarefa descendente, e considerando que a tarefa já foi alocada para um processador na primeira seção do cromossomo, a tarefa precedente precisa ser copiada para no mínimo $Succ(t_i) - 1$ processador, sendo $Succ(t_i)_i$ a quantidade de tarefas descendentes de uma tarefa t_i (JELODAR et al., 2006). Todavia, pode ser que a quantidade de processadores seja menor que a quantidade de tarefas descendentes, neste caso, a tarefa precedente pode ser copiada para todos os processadores disponíveis. Desta forma, a Expressão 14 pode ser utilizada para definir o comprimento de um cromossomo, sendo n a quantidade total de tarefas dispostas no grafo, m a quantidade de processadores disponíveis e n' as tarefas que podem ser duplicadas, isto é, tarefas que possuem custo de comunicação menor do que o atual custo de execução da tarefa.

$$n + \sum_{i=1}^{n'} \min(m - 1, \max(Succ(t_i) - 1, 0)) \tag{14}$$

O operador de recombinação aplicado à codificação FTE é a Recombinação de Ponto Único (SPX), onde uma separação é definida aleatoriamente, e as partes dividas são trocadas entre os pares da operação. O método é empregado por Jelodar et al. (2006) devido à sua simplicidade e por explorar a capacidade do cromossomo em manter-se em um comprimento fixo.

Já na aplicação da mutação, Jelodar et al. (2006) apresentam a possibilidade de que cada gene sofra uma variação de valor, configurando assim uma Mutação de Processador (PM).

4.2.3 Codificação de Intervalo (RE)

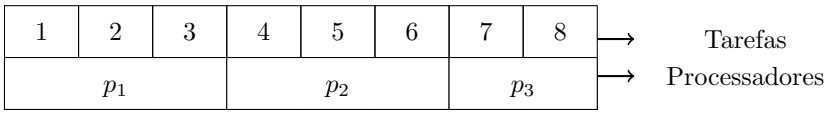
Um GA baseado em ponto de corte é a segunda técnica apresentado no trabalho de Hassan et al. (2015). Esse modelo diverge dos demais apresentados no texto, principalmente por sua codificação. Denominada como Codificação de Intervalo (RE[11]), a

[11] Do inglês, *Range Encoding*.

codificação adota duas linhas para representar o cromossomo: (i) uma linha determina uma ordem válida de acordo com as restrições de precedência; (ii) a segunda delimita a alocação de tarefas de acordo com um intervalo determinado por pontos de corte.

Inicialmente, para assinalar tarefas aos processadores, gera-se $m-1$ pontos de corte, de forma que $cp = \{cp_1, cp_2, ...cp_{m-1}\}$, onde m é a quantidade de processadores disponíveis. Por exemplo, o intervalo de tarefas, formado por t_0 até t_{cp_1}, aloca as tarefas do intervalo ao processador 1; o intervalo de tarefas, formado por t_{cp_1+1} até t_{cp_2}, aloca as tarefas do intervalo ao processador 2, assim sucessivamente. Conforme a definição de Hassan et al. (2015), uma subsequência válida de tarefas de tamanho aleatório é assinalada a um processador específico. Sobre o grafo da Figura 22(a), a Figura 35 apresenta um exemplo da codificação empregada em um cenário com três processadores.

Figura 35 – Codificação de Intervalo.



Fonte: do autor.

Utilizando como métrica o tempo total de execução, o GA proposto emprega, como operadores de modificação, uma Recombinação de Ponto Único (SPX) entre dois cromossomos e uma mutação baseada na troca entre dois pontos aleatórios (SM).

4.3 Representações baseadas em Alocação e Ordenação

Esta seção apresenta as representações que codificam uma combinação dos aspectos de ordenação e alocação. Primeiramente, a Subseção 4.3.1 apresenta a Codificação de Listas Topológicas, sendo essa uma representação pioneira utilizada em GAs. Em seguida, a Subseção 4.3.2 contextualiza a Codificação por Matrizes. Outro modelo popular, a Codificação de Alocação e Escalonamento é apresentada na Subseção 4.3.3. Por fim, a Codificação por Tuplas é apresentada na Subseção 4.3.4.

4.3.1 Codificação de Listas Topológicas (TLE)

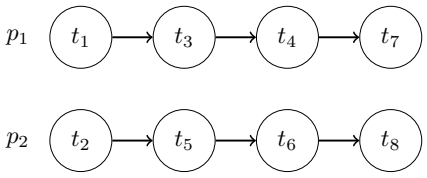
A primeira representação abordada nesta taxonomia é também a precursora utilizada no MTSP com algoritmos genéticos. Apresentada inicialmente por Hou, Hong e Ansari (1990), a codificação de listas topológicas (TLE[12]) distribui os nós de um grafo de tarefas

[12] Do inglês, *Topological List Encoding.*

em várias listas com tarefas computacionais, sendo que cada lista é associada a um processador individual. A ordenação de tarefas em cada lista é a maior restrição empregada por essa codificação.

A Figura 36(a) apresenta a TLE construída com base no grafo da Figura 22(a) e considerando um ambiente com dois processadores. Em aspecto computacional, a codificação pode ser implementada por meio de uma lista de valores inteiros com permutação (HOU; HONG; ANSARI, 1990), conforme demostrado na Figura 36(b). Todavia, é importante observar que nem todas as permutações alcançadas poderão gerar escalonamentos executáveis.

Figura 36 – Codificação de Lista Topológica.



(a) Cromossomo em TLE.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 5 & 4 & 7 & 6 & 8 & 1 & 2 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 & 4 & 7 \\ 3 & 4 & 7 & 1 \end{bmatrix} \begin{bmatrix} 2 & 5 & 6 & 8 \\ 5 & 6 & 8 & 2 \end{bmatrix}$$

(b) Exemplo com Permutações.

Fonte: adaptado de Hou, Ansari e Ren (1994).

A principal vantagem da TLE é a eliminação da necessidade de considerar as relações de precedência entre tarefas escalonadas para diferentes processadores (HOU; ANSARI; REN, 1994). Em outras palavras, é possível ignorar a relação inter-processadores até a etapa de decodificação do cromossomo. O cálculo do tempo de finalização é um exemplo de etapa que exige a decodificação. Porém, uma relação intra-processador, onde a relação de precedência entre tarefas ocorre dentro de um mesmo processador, deve ser mantida ao lidar com a codificação. Neste caso, existem variadas estratégias empregadas nesta ordem na organização das tarefas.

4.3.1.1 Ordenações

A seguir, são apresentadas técnicas utilizadas para determinar a ordenação das tarefas em cada lista na TLE. Os mecanismos listados não foram identificados como outras formas de representação devido às similaridades com o formato de lista topológicas. Portanto, o formato é o mesmo, mas a divergência ocorre no método de ordenação.

A primeira subseção apresenta a forma pioneira de ordenação, sendo essa baseada na altura das tarefas no DAG. Essa fórmula é expandida para duplicação de tarefas na subseção seguinte. Em seguida, é contextualizada uma melhoria na ordenação com o uso de dígrafos. Posteriormente, a implementação de outros índices de ordenação é apresentada nas subseções seguintes. Ao final, um cenário onde a ordenação pode ser substituída pela etapa de decodificação é apresentado.

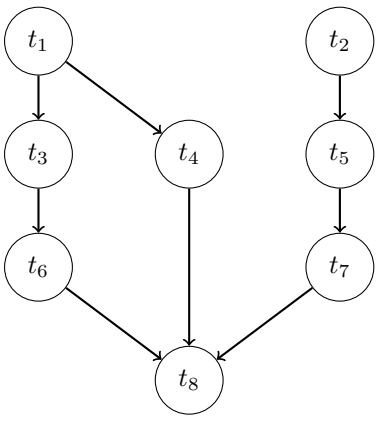
Ordenação Baseada em Altura

A ordem de execução das tarefas em cada processador é definida através da função *altura* (Expressão 15). Por exemplo, a lista topológica do processador 1 na Figura 36(a) estabelece sua ordem em: $altura(t_1) < altura(t_3) = altura(t_4) < altura(t_7)$. Contudo, Hou, Hong e Ansari (1990) destacam que uma representação de listas topológicas que controle a ordem das tarefas utilizando-se da função *altura* pode não alcançar um escalonamento ótimo. Desse modo, os autores propõem uma alteração inicial que define a função $altura'$: se a altura entre uma tarefa t_i e seus sucessores for maior que 1, a nova altura será um número aleatório (*rand*) entre altura(t_i) e $\max\{altura(t_j)\} - 1$, para todo $t_j \in succ(t_i)$; caso contrário, a altura não seria modificada. A definição de $altura'$ é apresentada na Expressão 16, de forma que $pred(t_i)$ representa as tarefas predecessoras à t_i e $succ(t_i)$ identifica as tarefas sucessoras à t_i. O DAG presente na Figura 37 apresenta um exemplo da aplicação de $altura'$, onde $et(t_i)$ representa o custo computacional (quantidade de instruções ou tempo necessário) para a executar uma tarefa t_i.

$$altura(t_i) = \begin{cases} 0 & \text{, se } pred(t_i) = \emptyset \\ 1 + \max\limits_{t_j \in pred(t_i)} \{altura(t_j)\} & \text{, caso contrário} \end{cases} \tag{15}$$

$$\begin{aligned} altura'(t_j) = rand \in (&\max\{altura'(t_i)\} + 1, \\ &\min\{altura'(t_k)\} - 1) \\ \text{Para todo } t_i \in pred(t_j) &\text{ e } t_k \in succ(t_j) \end{aligned} \tag{16}$$

Figura 37 – DAG adaptado à função $altura'$ (custo de comunicação arbitrário).



(a) DAG com 8 tarefas.

Tarefa	$et(t_i)$	$altura(t_i)$	$altura'(t_i)$
t_1	2	0	0
t_2	3	0	0
t_3	2	1	1
t_4	3	1	1 ou 2
t_5	2	1	2
t_6	3	2	2
t_7	2	2	2
t_8	1	3	3

(b) Relação entre $et(t_i)$, *altura* e $altura'$.

Fonte: adaptado de Tsujimura e Gen (1996).

Jezic et al. (1999) e Singh e Pillai (2014) são outros exemplos de trabalhos que utilizam essa codificação. Pillai et al. (2018) combinam a codificação de listas com nós que

também contempla valores de frequência utilizadas pelas tarefas em cenários onde a energia consumida também é avaliada. Além disso, utilizam a ordenação por altura com uma separação de grupos de tarefas com alturas em comum para, posteriormente, distribuí-las aos processadores. Zomaya, Ward e Macey (1999) implementam uma variação do cálculo da altura, de forma que a altura de uma tarefa, que possua tarefas predecessoras, é formada como o somatório de 1 mais todos os valores de altura.

Ordenação Baseada em Altura com Duplicação de Tarefas

TLE também pode ser estendida ao MTSP com duplicação de tarefas (TD[13]), em conformidade à adaptação proposta por Tsuchiya, Osada e Kikuno (1997). Nesse formato, as listas podem conter mais de uma cópia de cada tarefa, o que pode reduzir significativamente o tempo final de execução de um escalonamento com grande quantidade de comunicação entre tarefas. A Figura 38(a) contempla uma codificação de listas topológicas onde as tarefas t_1 e t_3 estão alocadas em ambos os processadores, p_1 e p_2. As tarefas utilizadas no exemplo da Figura 38(a) advêm do DAG apresentado na Figura 22(b). Em 38(a), o cromossomo c duplica as tarefas t_1 e t_3, tornando-se o cromossomo c'. Uma redução do tempo de conclusão do escalonamento pode ser observada no gráficos de Gantt dispostos na Figura 38(b). Considerando $tarefa(i, p)$ como a i-ésima tarefa da lista de processadores $p \in P$, a inclusão da duplicação não afeta a viabilidade do cromossomo desde que obedeça as três condições contextualizadas por Tsuchiya, Osada e Kikuno (1997):

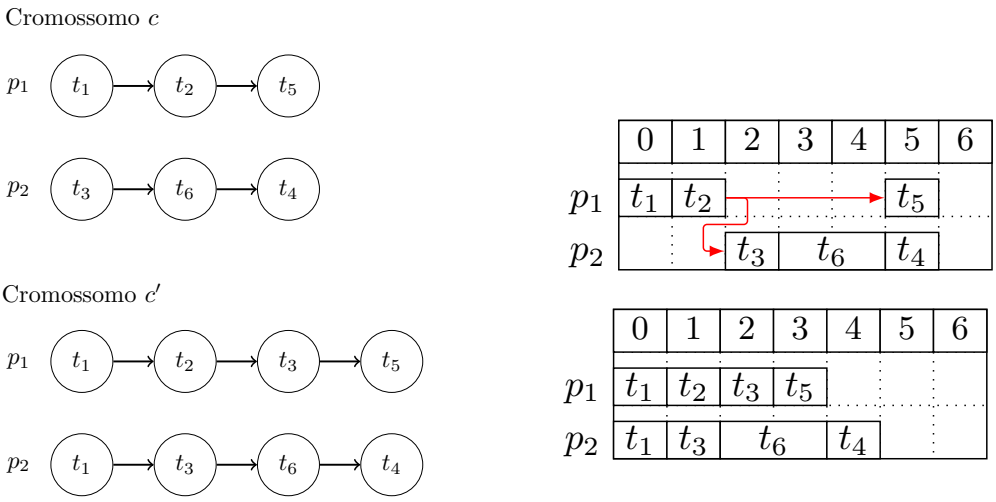
- **Condição 1:** Todas as tarefas estão presentes: $\forall t \in T$, há um $p \in P$ e um inteiro i tal que $t = tarefa(i, p)$;
- **Condição 2:** Não há duplicações de uma tarefa na mesma lista: $\forall p \in P$, se $i \neq j$, então $tarefa(i, p) \neq tarefa(j, p)$;
- **Condição 3:** As relações de precedência são mantidas: $\forall p \in P$, se $i < j$, então $tarefa(j, p) \succ tarefa(i, p)$.

Ordenação Baseada em Dígrafo

Apesar de garantir as restrições de precedência, controlar a ordenação das listas exclusivamente com a altura das tarefas pode apresentar algumas limitações, conforme observado por Corrêa, Ferreira e Rebreyend (1999). Considere o DAG disposto na Figura 39(a), em que cada tarefa possui custo computacional 1, com exceção da tarefa t_6, que

[13] Do inglês, *Task Duplication*.

Figura 38 – TLE aplicada com Duplicação de Tarefas.



(a) Duplicação de Tarefas. (b) Gráfico de Gantt de c e c'.

Fonte: adaptado de Tsuchiya, Osada e Kikuno (1997).

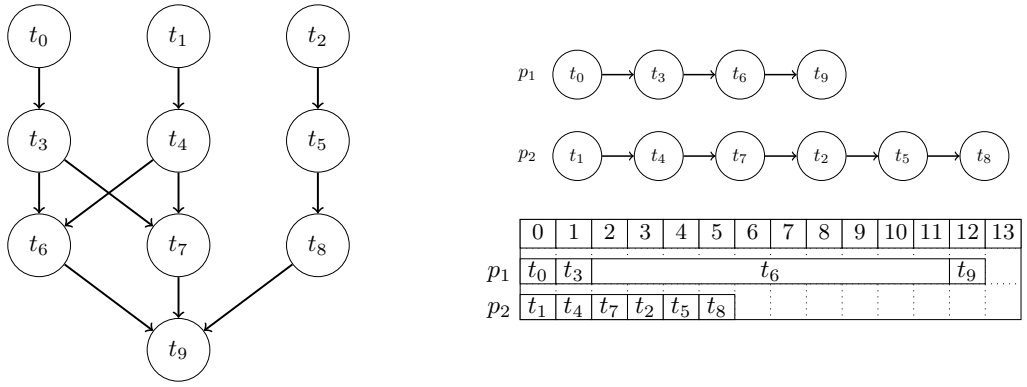
possui custo 10. Além de desconsiderar o custo de comunicação, cada tarefa do DAG possui o seguinte valor de altura (Expressão 17):

$$\begin{aligned} altura(t_0) = altura(t_1) = altura(t_2) = 1; \\ altura(t_3) = altura(t_4) = altura(t_5) = 2; \\ altura(t_6) = altura(t_7) = altura(t_8) = 3; \\ altura(t_9) = 4. \end{aligned} \tag{17}$$

Na sequência, a Figura 39(b) apresenta um escalonamento ótimo, de forma que a parte inferior da figura demonstra a relação entre tempo e custo computacional. Entretanto, apesar de coerente, a ordenação topológica disposta no cromossomo da Figura 39(b) não pode ser concebida com restrições de altura da codificação de Hou, Ansari e Ren (1994). No exemplo, as tarefas t_2 e t_5 quebram a regra ao apresentarem um valor de altura maior que as tarefas antecessoras assinaladas à p_2

Após o levantamento da limitação da altura das tarefas, Corrêa, Ferreira e Rebreyend (1999) elaboram outra forma de garantir a precedência e consequentemente, a factibilidade do escalonamento. O método emprega um DAG $D = (T, E)$ para verificar se uma transformação é possível. Dessa forma, em um escalonamento s, as tarefas presentes em um mesmo processador podem ter sua precedência definida através da Expressão 18. Todos os arcos contidos em $E(s)$ e que não estejam em E possuem custo de comunicação zero. Como definição de caminhos possíveis dos grafos, E^+ e $E^+(s)$ compõe os fechos transitivos de E e $E(s)$, respectivamente. Por fim, um DAG $(T, E(s))$ pode ser denotado

Figura 39 – Limitação da TLE com ordenação baseada em altura.



(a) DAG com 10 tarefas. (b) Cromossomo e Custo Computacional.

Fonte: adaptado de Corrêa, Ferreira e Rebreyend (1999), e Zhong e Yang (2003).

como $D(s)$.

$$\begin{aligned} E(s) = E \cup \{(t_{i_1}, t_{i_2}) | (t_{i_1}, t_{i_2}) \notin E, \\ p(t_{i_1}, s) = p(t_{i_2}, s) \\ r(t_{i_1}, s) = r(t_{i_2}, s) - 1\} \end{aligned} \tag{18}$$

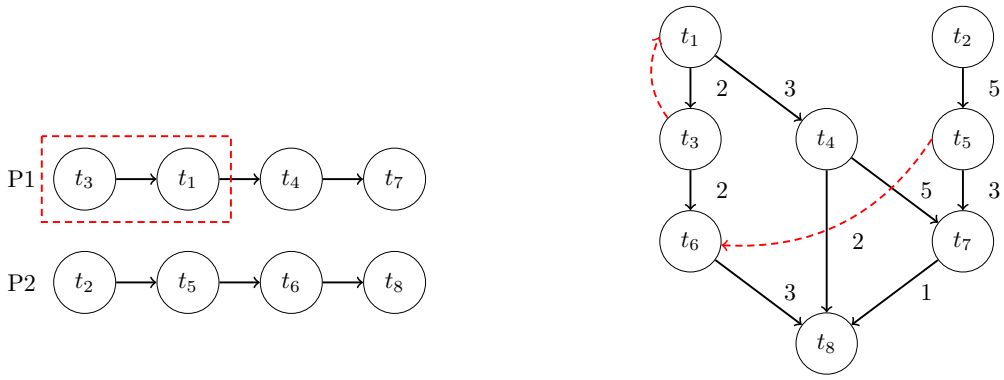
Assim, Corrêa, Ferreira e Rebreyend (1999) demonstram que um escalonamento s é factível se, e somente se, $D(s)$ é acíclico.

Um exemplo da ordenação é ilustrado na Figura 40. Inicialmente, a Figura 40(a) apresenta um cromossomo que possui anormalidade perante a ordenação topológica do DAG ao qual foi baseado (Figura 22(a)). O erro é caracterizado pela precedência das tarefas t_3 e t_1, conforme demarcado. Posteriormente, a Figura 40(b) apresenta o DAG $D(s)$ concebido pela Expressão 18. A irregularidade, presente em s, é destacada no caminho (t_3, t_1), uma vez que o arco configura um ciclo ao DAG original. Consequentemente, sendo $D(s)$ um grafo cíclico, s é um escalonamento infactível.

Ordenação Baseada em Índices

Outra adaptação da codificação é introduzida no trabalho de Golub e Kasapovic (2002). A estrutura de representação é idêntica a lista topológica, contudo, as tarefas não estão organizadas nas listas sobre seus valores de altura. A ordenação é efetuada, antecipadamente, de forma que todas as tarefas obtenham um índice que identifica sua precedência. Esse índices são calculados através de um algoritmo complementar. Considerando T como o conjunto de todas as tarefas, cada índice pode ser estabelecido como um número natural $[1...N]$ obtido através do cálculo e ordenação dos índices de ordem, *numero* e R. O Algoritmo 11 apresenta o método para obtenção desses índices.

Figura 40 – Validação das restrições de precedência com dígrafo.



(a) Erro de precedência em s.

(b) DAG $G'(V, E(s))$

Fonte: do autor.

Algoritmo 11 Ordenação topológica por índices – adaptado de Golub e Kasapovic (2002)

entrada: *numeros*, R;
saída: *numeros*, R;

```
1: para i = 1 até i <= |T| faça
2:    numeros[i] = numero_arestas_de_entrada[i];
3:    se numeros[i] == 0 então
4:       ordenar(R, i);
5:    fim se
6:    j = 1;
7:    enquanto R não está vazio faça
8:       l = primeiro_indice_da_ordem(R);
9:       indice[l] = j;
10:      j = j + 1;
11:      para todo k da lista_arestas_de_saida[l] faça
12:         numeros[k] = numeros[k] − 1;
13:         se numeros[k] == 0 então
14:            ordenar(R, k);
15:         fim se
16:      fim para
17:   fim enquanto
18:   i = i + 1;
19: fim para
```

Ordenação Baseada em Lista de Ordem de Execução

A codificação empregada por Zhong e Yang (2003) também é fortemente baseada no modelo TLE. A diferença é que as tarefas são divididas por uma lista de ordem de execução (TEOL[14]), que é gerada de acordo com o tempo de execução inicial das tarefas no escalonamento. Quanto antes uma tarefa é inserida na lista, mais cedo será seu momento

[14] Do inglês, *Task Executation Order List*.

de executar. Além disso, a construção de TEOL pode ser efetuada durante o cálculo do tempo de finalização (*makespan*), de forma que não seja necessário um tempo de computação extra para construí-la (ZHONG; YANG, 2003).

Ordenação Baseada em Índices Superior e Inferior

Também utilizando a codificação de listas, os trabalhos de Kaur, Chhabra e Singh (2010a), Kaur, Chhabra e Singh (2010b), Singh (2012) propõem a utilização de índices *t-level* ou *b-level* (Subseção 2.1.1) como forma de ordenação. Considerando o DAG, o índice *t-level* indica o comprimento do mais longo caminho de uma tarefa de entrada até uma tarefa t_i, enquanto que o índice *b-level* define o mais longo caminho entre uma tarefa t_i e uma tarefa de saída. Além disso, esses índices ainda podem ser combinados com heurísticas, tais como a min-min ou tempo de execução mínimo.

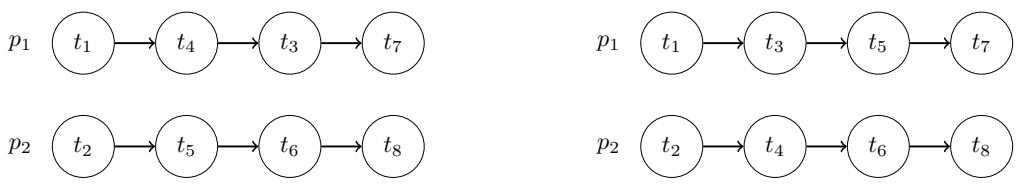
A Tabela 7 apresenta os índices *b-level* e *t-level* calculados e ordenados sobre o grafo de tarefas da Figura 22(a), sendo que $et(t_i)$ indica o custo computacional de uma tarefa t_i. Em adição, a Figura 41 apresenta dois cromossomos formados e ordenados através do índices *b-level*, Figura 41(a), e *t-level*, Figura 41(b).

Tabela 7 – Relação entre os tempos de execução e conclusão, índices inferior e superior, e ordenação de índices.

T	$et(t_i)$	*b-level*	*t-level*	$et(t_i)$ ordenado	*b-level* ordenado	*t-level* ordenado
t_1	3	20	0	t_3	t_1	t_1
t_2	2	20	0	t_2	t_2	t_2
t_3	1	14	5	t_4	t_4	t_3
t_4	2	13	6	t_6	t_5	t_4
t_5	3	12	7	t_7	t_3	t_5
t_6	2	9	8	t_1	t_6	t_6
t_7	2	7	13	t_5	t_7	t_7
t_8	4	4	16	t_8	t_8	t_8

Fonte: adaptado de Kaur, Chhabra e Singh (2010a) e Kaur, Chhabra e Singh (2010b).

Figura 41 – Codificação por lista topológica com índices.



(a) Resolução por *b-level*(t_i). (b) Resolução por *t-level*(t_i)

Fonte: adaptado de Kaur, Chhabra e Singh (2010a) e Kaur, Chhabra e Singh (2010b).

Ordenação Realizada na decodificação

Dependendo dos operadores de recombinação e mutação empregados, as tarefas dispostas nas listas de um cromossomo podem não apresentar uma ordenação explicitamente válida. Daoud e Kharma (2011) é um exemplo de trabalho que adota a codificação de listas com essa variação, isto é, apesar de poder possuir tarefas distribuídas sem seguir explicitamente o grafo de tarefas, o cromossomo pode representar um escalonamento válido através do processo de decodificação. Portanto, cromossomos que à priori seriam definidos como inválidos, podem ser explorados.

4.3.1.2 Recombinações

Nesta seção são contextualizadas as diversas recombinações empregadas na TLE. A primeira subseção apresenta uma recombinação pioneira, sendo essa baseada na altura das tarefas. As subseções seguintes contextualizam modelos com o uso de seleção de tarefas específicas, utilização de múltiplos cortes para troca de material, ordenações de dígrafos, permutações, fusões, formas de ordenação, expansão do ponto de corte e combinação de máscara com permutações.

Recombinação de Ponto Único (SPX)

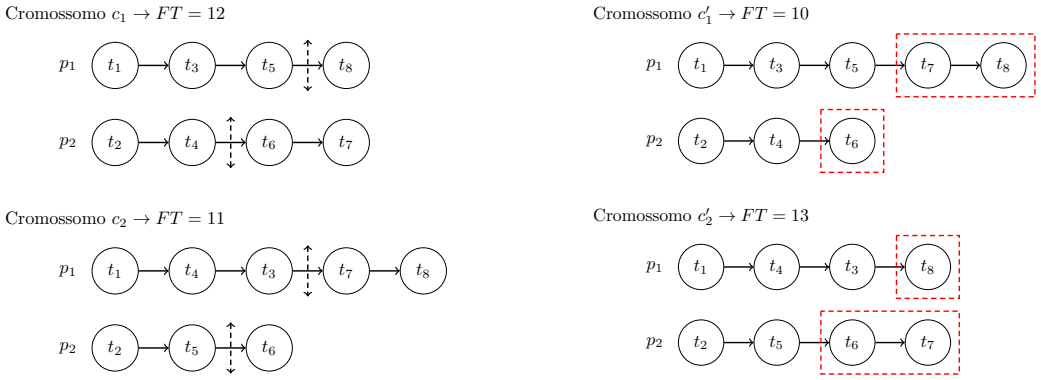
Semelhante a recombinação de um ponto, a recombinação de ponto único (SPX[15]) é implementada sobre a definição de pontos de corte entre as listas de cada processador. A operação promove a troca de tarefas entre as listas (processadores) de cada cromossomo e pode ser estendida para ambientes com P processadores.

A Figura 42 ilustra a aplicação da recombinação baseada em altura em dois cromossomos construídos sob o grafo da Figura 22(a), denominados c_1 e c_2. Os pontos de corte adicionados aos cromossomos, Figura 42(a), estão posicionados entre diferentes valores de altura de cada tarefa, por exemplo, no cromossomo c_1, $altura(t_5) \neq altura(t_8)$ e $altura(t_4) \neq altura(t_6)$. Após a definição dos pontos de cortes, a Figura 42(b) demonstra a criação de novos cromossomos, c'_1 e c'_2, originados pela troca das tarefas alocadas em cada processador. Em consequência da recombinação, o cromossomo c'_1 (Figura 42(b)) contempla um tempo de finalização menor que seus descendentes.

Contudo, para a geração de escalonadores válidos, isto é, onde todas as tarefas são alocadas sem repetição, Hou, Hong e Ansari (1990) observam duas propriedades essenciais à escolha dos pontos de corte, sendo essas: (i) as tarefas próximas ao ponto de corte possuem alturas diferentes; (ii) as alturas de todas as tarefas em frente do ponto de corte são as mesmas. Partindo dessas propriedades, Hou, Ansari e Ren (1994) apresentam

[15] Do inglês, *Single Point Crossover*.

Figura 42 – Recombinação de altura na TLE.



(a) Cromossomos c_1 e c_2. (b) Descendentes c'_1 e c'_2.

Fonte: adaptado de Hou, Hong e Ansari (1990).

o exemplo ilustrado pela Figura 43. Inicialmente, na Figura 43(a), os pontos de corte atendem à seguinte restrição:

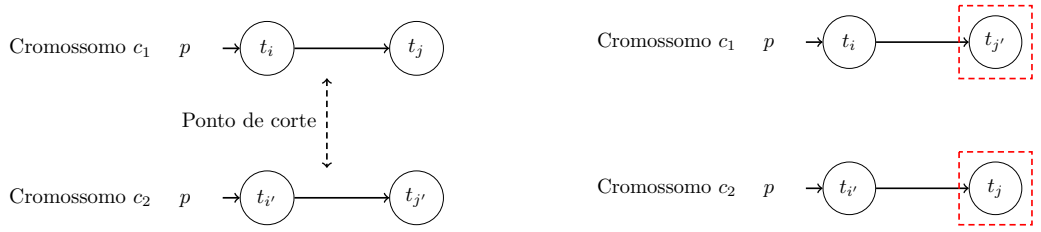
$$altura(t_i) < altura(t_j), altura(t_{i'}) < altura(t_{j'}), altura(t_i) = altura(ti') \tag{19}$$

Após a recombinação proposta pelos pontos de corte, a Figura 43(b) apresenta a seguinte relação válida:

$$altura(t_i) < altura(t_{j'}), altura(t_{i'}) < altura(t_j) \tag{20}$$

Desde que as tarefas, que tenham altura maior que $altura(t_i)$, sejam trocadas entre cromossomos, nenhuma tarefa é excluída ou duplicada (HOU; ANSARI; REN, 1994). Em uma ambiente onde TD não é permitida, as propriedades de completude e exclusividade são preservadas. Implementando as propriedades supracitadas, as operações realizadas na recombinação baseada em altura estão enumeradas no Algoritmo 12.

Figura 43 – Demonstração da restrição na Recombinação de Altura.



(a) Ponto de corte. (b) Relação Válida.

Fonte: adaptado de Hou, Ansari e Ren (1994).

O trabalho de Jezic et al. (1999) aplica uma variação dessa recombinação ao selecionar todos os pontos de corte com valores de altura iguais. Corrêa, Ferreira e Rebreyend (1999)

Algoritmo 12 Recombinação de Altura – adaptado de Hou, Hong e Ansari (1990)

entrada: cromossomos c_1 e c_2
saída: cromossomos c'_1 e c'_2

{Seleção de pontos de recombinação:}
1: Aleatoriamente, gere um número, x, entre 0 e a altura máxima do grafo de tarefas;
{Laço para cada processador:}
2: Para cada processador p_i nos cromossomos c_1 e c_2, faça o passo 3;
{Encontrar pontos de recombinação:}
3: Encontre as tarefas consecutivas t^i_j e t^i_k em cada processador p_i tal que $x = altura(t^i_j) < altura(t^i_k)$, $altura(t^i_k)$ seja a mesma para todo i;
{Laço para cada processador:}
4: Para cada processador p_i nos cromossomos c_1 e c_2, faça o passo 5;
{Recombinação:}
5: Troque as extremidades de tarefas dos cromossomos c_1 e c_2 para cada processador p_i.

(por comparação), Kaur, Chhabra e Singh (2010a), Kaur, Chhabra e Singh (2010b), Singh (2012), Singh e Pillai (2014), Singh e Singh (2012) e Pillai et al. (2018) são outros exemplos de trabalhos que utilizam a recombinação de um ponto único.

Recombinação de Tarefas-Alvo (TTX)

Com a formação de um conjunto de tarefas, denominado tarefas-alvo, e inicialmente proposta por Woo et al. (1997), a Recombinação de Tarefas-Alvo (TTX[16]) utiliza as relações de precedência entre uma tarefa do grafo e suas respectivas tarefas sucessoras imediatas. Inicialmente, uma tarefa t_i é aleatoriamente selecionada do grafo de tarefas usado na entrada. Posteriormente, a formação do conjunto é dado por $tarefas\_alvo = t_i \cup Succ(t_i)$, onde $Succ(t_i)$ é o conjunto de tarefas sucessoras imediatas de t_i.

O Algoritmo 13 sumariza as instruções do TTX. A execução da função $extrair\_tarefas$ remove as tarefas alvo do processador (p_u) de cada cromossomo informado. Ao ser finalizada, $extrair\_tarefas$ retorna as tarefas removidas ao conjunto de tarefas extraídas. Em contrapartida, a função $inserir\_tarefas$ distribui randomicamente as tarefas para um processador de algum cromossomo obedecendo as relações de precedência. Um exemplo da aplicação do algoritmo é ilustrado na Figura 44 onde a tarefa selecionada é t_4.

Para que uma tarefa t_i seja inserida em um processador p_u com a função $inserir\_tarefas$, a respectiva tarefa precedente t_p e a tarefa sucessora t_f devem obedecer a seguinte restrição (WOO et al., 1997):

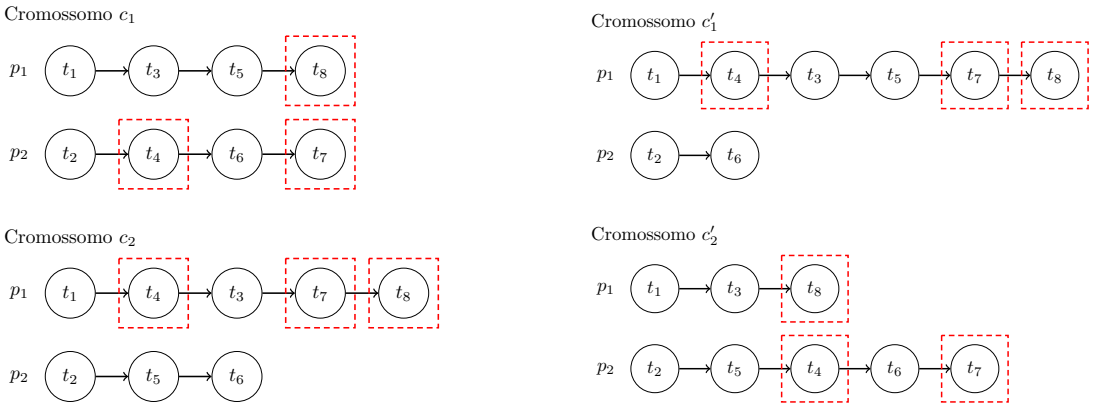
$$altura(t_p) \leq altura(t_i) \leq altura(t_f) \tag{21}$$

[16] Do inglês, *Target Task Crossover*.

Algoritmo 13 Recombinação baseada em tarefas-alvo Woo et al. (1997)

entrada: Cromossomos c_1 e c_2;
saída: Cromossomos c'_1 e c'_2;
1: Aleatoriamente, selecione uma tarefa t_i do grafo de entrada;
2: $tarefas\_alvo \leftarrow t_i \cup Succ(t_i)$;
3: **para** u **até** P **faça**
4: $\quad tarefas\_extraidas\_c_1 \leftarrow extrair\_tarefas(c_1, p_u, tarefas\_alvo)$;
5: $\quad tarefas\_extraidas\_c_2 \leftarrow extrair\_tarefas(c_2, p_u, tarefas\_alvo)$;
$\quad$ {Gerando c'_1 e c'_2}
6: $\quad inserir\_tarefas(c_1, p_u, tarefas\_extraidas\_c_2)$;
7: $\quad inserir\_tarefas(c_2, p_u, tarefas\_extraidas\_c_1)$;
8: **fim para**
9: **retorne:** c'_1 e c'_2;

Figura 44 – Recombinação de Tarefas-Alvo.



(a) Seleção das tarefas em c_1 e c_2. (b) Aplicação do operador.

Fonte: adaptado de Woo et al. (1997).

Recombinação Multiponto (MPX)

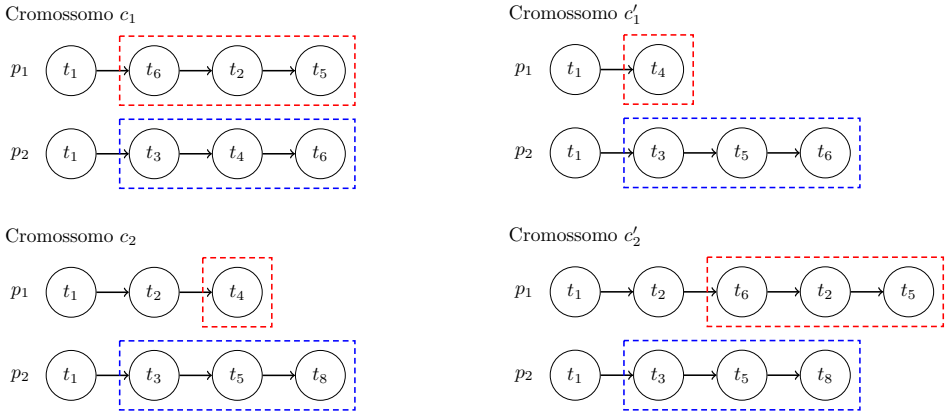
Segundo as definições de Tsuchiya, Osada e Kikuno (1997), inicialmente, a Recombinação Multiponto(MPX[17]) gera um número inteiro aleatório x $(0 \leq x \leq maxaltura(t_i)|t_i \in T)$ como índice para a seleção de pontos de corte. Posteriormente, para cada lista associada ao processador p, nos cromossomos c_i e c_{i+1}, um ponto de corte é selecionado, de forma que possibilite a divisão da lista em duas partes. A definição do intervalo do ponto de corte pode ser definida como: tarefas na primeira parte possuem valor de altura menor que x, em contrapartida, a primeira tarefa da segunda parte deverá possuir valor de altura maior ou igual a x. A seleção de pontos de corte baseada por altura previne que a recombinação "quebre" informações hereditárias úteis ao reduzir a probabilidade da execução de operações de ajuste (TSUCHIYA; OSADA; KIKUNO, 1997). Ao final da operação, as

[17] Do inglês, *Multipoint Crossover*.

partes inferiores são trocadas. Cromossomos submetidos ao MPX podem gerar soluções incompletas ou baseadas em TD.

A Figura 45(a) apresenta um exemplo do operador. Para os processadores, p_1 e p_2, considere como índices de corte as alturas 2 e 1, respectivamente. A recombinação dos cromossomos, c_1 e c_2, é efetuada após a definição dos quatro pontes de corte. Ao final, a Figura 45(b) apresenta os novos cromossomos, denominados c'_1 e c'_2.

Figura 45 – Recombinação Multicorte de altura com Duplicação de Tarefas.



(a) Índices $p_1 = 2$ e $p_2 = 1$. (b) Aplicação da Recombinação.

Fonte: adaptado de Tsuchiya, Osada e Kikuno (1997).

A diferença em relação à recombinação baseada em altura (Subseção 4.3.1.2) está na determinação dos índices responsáveis pelos cortes. Na recombinação baseada em altura, um único ponto de corte c é definido e utilizado para todas as listas dos cromossomos, isto é, atribuído a todo p_i. O intervalo também é diferente, uma vez que as tarefas anteriores à c possuem mesmo valor de altura, enquanto que as tarefas posteriores à c possuem maior valor de altura.

Recombinação de Dígrafo (DX)

Assim como os demais operadores na TLE, a Recombinação de Dígrafo (DX[18]) é executada sobre um par de cromossomos, c_1 e c_2. O operador também é semelhante a recombinação de um ponto, de forma que a divisão efetuada pelo corte cria conjuntos de tarefas independentes. Essa propriedade é denominada por Corrêa, Ferreira e Rebreyend (1999) como um conjunto fechado de tarefas, tal como um subconjunto V', em que $t \in V'$, então as tarefas predecessoras de t também pertencem a V'. Desse modo, o operador deve definir dois subconjuntos, V_1 e V_2, de tal forma que as tarefas dispostas em V_2 não possuam dependências com as tarefas dispostas em V_1. Portanto, o subconjunto V_1 deve ser um conjunto fechado de tarefas.

[18] Do inglês, *Digraph Crossover*.

Além disso, uma vez que as tarefas não são ordenadas por sua altura (Subseção 4.3.1.1), Corrêa, Ferreira e Rebreyend (1999) utilizam a Expressão 18 para construir um subconjuto que obedece a condição de D(s) ser acíclico (Subseção 4.3.1.1). Sendo assim, as dependências do dígrafo, bem como as dependências dos cromossomos, c_1 e c_2, são representadas no subconjunto formado pelo dígrafo $(\tau, E(s_1) \cup E(s_2))$, sendo T o conjunto de todas as tarefas do DAG e $T = \tau$. Dessa forma, Corrêa, Ferreira e Rebreyend (1999) executam os seguintes passos enquanto $T \neq \emptyset$.

1. Selecione aleatoriamente uma tarefa $t_i \in T$ e $V = V_j$, $j = 1$ ou 2.

2. Se $V = V_1$ então

$$\begin{aligned} V_1 \leftarrow V_1 \cup \{t_i\} \cup \\ \{t'_i : t'_i \in T \text{ e} \\ (t'_i, t_i) \in (E^+(s_1) \cup E^+(s_2))\}, \end{aligned} \tag{22}$$

   caso contrário

$$\begin{aligned} V_2 \leftarrow V_2 \cup t_i \cup \\ \{t'_i : t'_i \in T \text{ e} \\ (t_i, t'_i) \in (E^+(s_1) \cup E^+(s_2))\}. \end{aligned} \tag{23}$$

3. Remova todas as tarefas inseridas em V_1 ou V_2 de T.

Conforme a definição de Corrêa, Ferreira e Rebreyend (1999), na segunda instrução (2), todas as tarefas predecessoras de t_i e a própria t_i são inseridas em V_1. Em paralelo, na terceira instrução (3), t_i e todas suas tarefas sucessoras são inseridas em V_2. Portanto, V_1 e V_2 correspondem às partições requeridas quando $T = \emptyset$.

Uma vez que as tarefas presentes em V_1 representam o segmento à esquerda, e as tarefas em V_2 representam o segmento à direita, os cromossomo descendentes, c'_1 e c'_2 podem ser formados trocando os segmentos à direita, isto é, semelhantemente às permutações geradas na recombinação de um ponto. Desse modo, seguindo a definição de Corrêa, Ferreira e Rebreyend (1999), para gerar um descendente c'_1, as tarefas em V_1 são escalonadas conforme c_1, enquanto que as tarefas dispostas em V_2 são escalonadas utilizando uma heurística de lista com as seguintes regras:

- ❑ R_1**:** Selecione a tarefa t_i possuindo a menor classificação $r(t_i, c_2)$. Em caso de empate, selecionar uma aleatoriamente;

- ❑ R_2**:** Selecione o processador $p(t_i, c_2)$.

Corrêa, Ferreira e Rebreyend (1999) alertam que, usando esse método, a ordem de tarefas pertencentes em V_2 é mesma que c'_1 e c_2, entretanto suas introduções podem variar. De forma análoga, o cromossomos descendente c'_2 é formado ao manter as tarefas

de V_1 escalonadas conforme c_2 e considerando c_1 para formar as tarefas pertencentes ao segmento V_2.

Os autores também apresentam uma variação desse operador. Após a definição dos segmentos V_1 e V_2, na geração do cromossomos descendente c'_1, no momento da troca das partes à direita, o operador opta por escalonar as tarefas restantes em V_2 através de um algoritmo de lista executado sobre $D(c_2)$. A heurística acoplada ao operador é denominada como Primeiro Sucessor mais Imediato (MISF[19]), sendo composto pelas seguintes regras (CORRÊA; FERREIRA; REBREYEND, 1999):

- **R_1:** Calcule o tempo mínimo de introdução de cada tarefa livre. Em seguida, selecione a tarefa, t_i, com o menor tempo de inicialização. Em caso de diversas possibilidades, selecione a tarefa com a maior quantidade de sucessoras. Em caso de diversas possibilidades, selecionar aleatoriamente. O tempo mínimo de introdução é calculado sobre as restrições de precedência e sobre as tarefas previamente escalonadas;

- **R_2:** Selecione aleatoriamente um processador dentre os processadores que a tarefa t_i pode ser executada o mais rápido possível.

Ao final, o cromossomos descendente c'_2 pode ser formado da mesma forma, sendo as tarefas dispostas em V_2 escalonadas sobre as restrições de $D(c_1)$.

Recombinação Parcialmente Combinada (PMX)

Em codificações baseadas em permutações, a Recombinação Parcialmente Combinada (PMX[20]) é uma das mais populares. Implementada por Zomaya, Ward e Macey (1999) sobre a TLE, PMX é aplicado em cromossomos unidimensionais. Portanto, o formato bidimensional adotado pela codificação de listas é convertido a uma versão unidimensional. Zomaya, Ward e Macey (1999) aplicam essa recombinação de duas formas diferentes: (i) a subcadeia transferida entre os cromossomos pais tem tamanho fixo ou (ii) dois pontos de corte são selecionados aleatoriamente para definir a subcadeia transferida.

Segundo a definição de Zomaya, Ward e Macey (1999), após a definição da subcadeia (ou janela) nos cromossomos pais, independente da forma de escolha dos pontos de corte, a primeira tarefa livre dentro da subcadeia do primeiro pai é encontrada. A tarefa de número correspondente é encontrada no segundo cromossomo (tarefa i do segundo cromossomo, isto é, t_{i2}). Se esta tarefa for encontrada na subcadeia, então o primeiro par foi encontrado. Se a mesma for encontrada dentro da subcadeia e na mesma posição do primeiro cromossomo, ambas são marcadas como prontas e a próxima tarefa livre é encontrada. Se a tarefa for encontrada dentro da subcadeia, mas em uma posição diferente da

[19] Do inglês, *Most Immediate Successors First*.
[20] Do inglês, *Partially Matched Crossover*.

tarefa do primeiro cromossomo, então a tarefa no primeiro cromossomo é marcada como o primeiro par (a qual está na mesma posição que t_{i2}). Para cada nova tarefa parceira encontrada, o processo é repetido enquanto esta possuir tarefas correspondentes fora da subcadeia no segundo cromossomo. Para encontrar a segunda tarefa parceira (uma tarefa do segundo cromossomo), todo o processo é repetido. Após a formação de um par, as tarefas parceiras têm seus respectivos valores trocados, sendo o processo completo repetido até que nenhuma tarefa livre reste na subcadeia. Por fim, as subcadeias são trocadas entre os cromossomos pais e, para a construção de soluções válidas, quando a recombinação é finalizada, o operador reordena e reconverte os cromossomos ao formato bidimensional. O Algoritmo 14 enumera as instruções utilizadas no PMX.

Recombinação Cíclica (CX)

A recombinação cíclica (CX[21]) é outro operador geralmente utilizado em codificações baseadas em permutações. Zomaya, Ward e Macey (1999) aplicam o CX com duas variações, sendo a primeira a constante escolha do primeiro gene pra começar a aplicação do operador, enquanto que a segunda é uma definição aleatória de qual gene começará o ciclo.

Semelhante ao PMX aplicado na TLE, dois pares de cromossomos são selecionados e convertidos para versões unidimensionais. Após as conversões, uma das variações supracitadas é acionada para começar a aplicação, consequentemente, selecionando a primeira tarefa do ciclo. Em seguida, seguindo a definição de Zomaya, Ward e Macey (1999), a primeira tarefa selecionada é identificada como t_{i1} (presente no primeiro cromossomo). Essa tarefa é marcada como concluída. A tarefa correspondente presente no segundo cromossomo é localizada, t_{i2}. Esta tarefa é marcada como concluída. A tarefa presente no primeiro cromossomo na mesma posição que t_{i2} é marcado como concluída. A tarefa correspondente à tarefa no primeiro cromossomo (t_{i1}) é localizada no segundo cromossomo (t_{j2}). Essa tarefa é marcada como concluída. Dessa forma, o processo é repetido até que a tarefa do inicio da aplicação seja localizada novamente (t_s1). Tarefas presentes no ciclo (aquelas que foram demarcadas como concluídas) permanecem na mesma posição; as demais tarefas não demarcadas são trocadas entre os cromossomos pais, por exemplo, a tarefa na posição 2 do primeiro cromossomo, t_{21}, é trocada com a tarefa na posição 2 do segundo cromossomo, t_{22} (ZOMAYA; WARD; MACEY, 1999). Ao final, os cromossomos são reordenados e reconvertidos ao formato bidimensional. A descrição da recombinação cíclica supracitada é sumarizada no Algoritmo 15.

[21] Do inglês, *Cycle Crossover*.

Algoritmo 14 Recombinação Parcialmente Combinada – adaptado de Zomaya, Ward e Macey (1999)

entrada: *tipo*, $escalonamento_1$, $escalonamento_2$;
saída: $escalonamento_1$, $escalonamento_2$;

```
armazenar_cadeia1(escalonamento1); armazenar_cadeia2(escalonamento2);
definir pontos de corte, ponto1 e ponto2, segundo o tipo;
laço
  obtêm a próxima tarefa livre na cadeia1;
  laço
    encontrar_tarefa(2, tarefa);
    se posição da tarefa fora dos pontos de corte então
      tarefa é a primeira tarefa parceira; sair do sublaço;
    senão se posição da tarefa conforme a atual então
      obtêm a próxima tarefa livre na cadeia1;
      se nenhuma tarefa livre for encontrada então
        sair do laço principal;
      fim se
    senão se posição da tarefa entre os pontos de corte então
      se loop interno encontrado então
        obtenha a próxima tarefa livre na cadeia1;
        se nenhuma tarefa livre for encontrada então
          sair do laço principal;
        senão
          demarque tarefa como concluída;
          próxima tarefa é tarefa na cadeia1 na atual posição na cadeia2;
        fim se
      fim se
    fim se
  fim laço
  obtenha a primeira tarefa livre na cadeia2;
  laço
    encontrar_tarefa(1, tarefa);
    se posição da tarefa fora dos pontos de corte então
      tarefa é o segunda tarefa parceira; sair do sublaço;
    senão se posição da tarefa entre os pontos de corte então
      demarque a tarefa como concluída;
      próxima tarefa é a tarefa na cadeia2 na atual posição da cadeia1;
    fim se
  fim laço
  troque as tarefas; obtenha a próxima tarefa da recombinação;
  se não há mais tarefas livres então
    saia do laço;
  fim se
fim laço
PMX_troca(ponto1, ponto2);
reordenar(1, escalonamento1); reordenar(2, escalonamento2);
restaurar(1, escalonamento1);
```

Algoritmo 15 Recombinação Cíclica – adaptado de Zomaya, Ward e Macey (1999)

entrada: *tipo*, $escalonamento_1$, $escalonamento_2$;
saída: $escalonamento_1$, $escalonamento_2$;

1: $armazenar\_cadeia_1(escalonamento_1)$; $armazenar\_cadeia_2(escalonamento_2)$;
{Encontrando o ponto inicial}
2: **se** tipo = 1 **então**
3: ponto = 0;
4: **senão se** tipo = 2 **então**
5: ponto = aleatório;
6: **fim se**
7: obtenha a primeira $tarefa$;
8: **laço**
9: marque a $tarefa$ como pronta;
10: pegue a $tarefa$ na $cadeia_1$ usando a atual posição na $cadeia_2$;
11: **se** $tarefa$ atual = primeira $tarefa$ **então**
12: saia do loop;
13: **fim se**
14: obtenha a próxima $tarefa$;
15: **fim laço**
{Realizando a recombinação}
16: **para** $i = 0$ **até** quantidade de tarefas **faça**
17: **se** tarefa não está no ciclo **então**
18: troque a tarefa da $cadeia_1$ com a tarefa do $cadeia_2$;
19: troque a tarefa da $cadeia_2$ com a tarefa da $cadeia_1$;
20: **fim se**
21: **fim para**
22: $reordenar(1, escalonamento_1)$; $reordenar(2, escalonamento_2)$;
23: $restaurar(1, escalonamento_1)$; $restaurar(2, escalonamento_2)$;

Recombinação de Fusão (FX)

Proposto por Golub e Kasapovic (2002), a recombinação de fusão (FX[22]) gera um único filho, sendo aplicado quando os cromossomos pais selecionados são diferentes. Ao selecionar dois cromossomos aleatórios, a construção de um novo filho é efetuada através de uma varredura completa sobre todo o conjunto de tarefas, de forma que cada tarefa seja anexada, aleatoriamente, ao filho, orientando-se pelas posições originais de seus pais. Se a tarefa t_i está presente no mesmo processador de ambos os pais, isto é, na mesma lista, a tarefa t_i é copiada e anexada na mesma posição no novo cromossomo (GOLUB; KASAPOVIC, 2002). Caso contrário, a tarefa t_i copiada ao filho será posicionada, aleatoriamente, em conformidade com algum dos pais selecionados. A restrição de precedência deve ser obedecida, sendo assim implementada pelo resultado do algoritmo de ordenação topológica. Considerando T como a quantidade total de tarefas, o Algoritmo 16 enumera as instruções do FX.

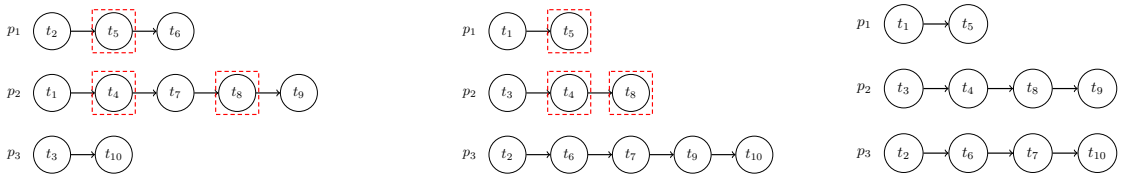
[22] Do inglês, *Fusion Crossover*.

Algoritmo 16 Recombinação de Fusão – adaptado de Golub e Kasapovic (2002)

entrada: Cromossomos pais c_1 e c_2;
saída: Cromossomo descendente c;
1: **para** $i = 1$ **até** $i \leq T$ **faça**
2: **se** ($t_i \in P$ está presente em ambos os pais) **então**
3: Copie t_i para o mesma mesma posição no novo cromossomo;
4: **senão**
5: Copie aleatoriamente t_i para c em uma das listas baseada em seus pais;
6: **fim se**
7: $i = i + 1$;
8: **fim para**
9: **retorne:** c;

A Figura 46 apresenta um exemplo da recombinação por fusão. Os cromossomos selecionados c_1 e c_2, respectivamente Figuras 46(a) e 46(b), efetuam a recombinação ao copiar tarefas ao cromossomo filho (Figura 46(c)). A cópia de tarefas submete-se a duas situações: (i) tarefas comuns aos processadores de ambos os pais são copiadas para a mesmo processador no filho, por exemplo, as tarefas t_4, t_5 e t_8; (ii) tarefas que estejam em listas distintas, entre os cromossomos selecionados, são alocadas, aleatoriamente, em algum dos processadores listados pelos pais.

Figura 46 – Recombinação de Fusão.



(a) Cromossomos c_1. (b) Cromossomo c_2. (c) Cromossomo filho.

Fonte: adaptado de Golub e Kasapovic (2002).

Recombinação por Lista de Ordem de Execução (TEOLX)

A Recombinação por Lista de Ordem de Execução (TEOLX[23]), proposta por Zhong e Yang (2003), utiliza o TEOL (Subseção 4.3.1.1) para dividir o par de cromossomos selecionados. Posteriormente, as partes dispostas à esquerda do corte são mantidas, enquanto que as partes à direita são reescalonadas com a utilização de uma heurística sobre as restrições de precedência do cromossomo adjacente. As instruções para execução da recombinação estão enumeradas no Algoritmo 17, onde c_1 e c_2 representam o par de cromossomos.

[23] Do inglês, *Task Execution Order List Crossover*.

Algoritmo 17 Recombinação por Lista de Ordem de Execução – adaptado de Zhong e Yang (2003)

entrada: Cromossomos c_1 e c_2;
saída: Cromossomos c'_1 e c'_2;
1: Pegue uma tarefa t_i aleatoriamente;
2: Particione as tarefas nos conjuntos V_1 e V_2, de forma que:
3: **para todo** tarefa t_j em V **faça**
4: **se** t_j está em frente de t_i em $TEOL(c_1)$ ou $TEOL(c_2)$ **então**
5: Coloque t_j em V_1;
6: **senão**
7: Coloque t_j em V_2;
8: **fim se**
9: **fim para**
10: **para todo** tarefa t_j em V **faça**
11: **se** t_j está em V_2 **então**
12: Apague t_j em c_1 e c_2;
13: **fim se**
14: **fim para**
15: Reescalone as tarefas em V_2 para c_1 e c_2 usando o algoritmo de ordenação de acordo com as ordens de execução em c_2 e c_1.

Recombinação de Dois Pontos (TPX)

Além de aplicarem o SPX, Kaur, Chhabra e Singh (2010a), Kaur, Chhabra e Singh (2010b), Singh e Singh (2012), Singh (2012) e Singh e Pillai (2014) também propõem a Recombinação de Dois Pontos (TPX[24]). Neste modelo, dois pontos de cortes dividem as diversas listas de processadores, definindo sublistas que serão trocadas entre os pares de cromossomos. Com a ordenação das listas definidas por índices de *b-level* ou *t-level*, os pontos de corte são posicionados de forma a não violar restrições de precedência. A Figura 47 ilustra um exemplo da recombinação de dois pontos.

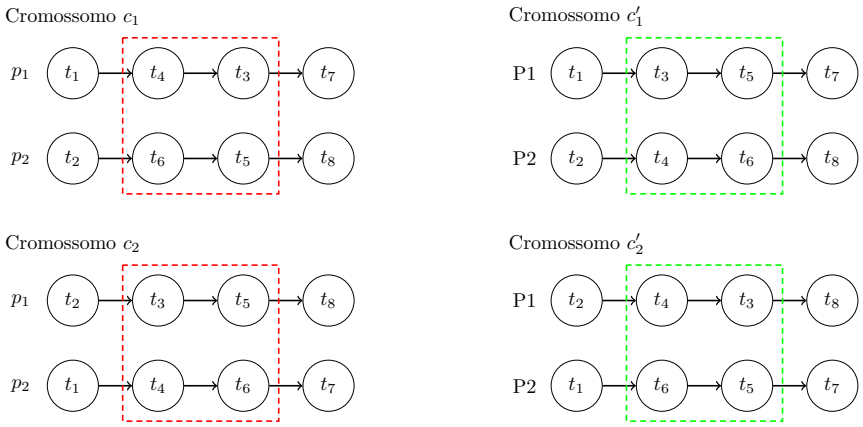
Recombinação de Máscara (MX)

Aplicado em pares de cromossomos, a Recombinação de Máscaras (MX[25]) gera novos cromossomos transmitindo e combinando segmentos de ordem relativa de execução de tarefas. Apresentada no trabalho de Daoud e Kharma (2011), a MX gera cromossomos que são amparados pelo mecanismo de decodificação de forma que, ao percorrer o grafo de tarefas para construir o escalonamento, mesmo tarefas posicionadas em posições inválidas na codificação formam escalonamentos válidos.

Inicialmente, o operador define uma subcadeia ao sortear dois pontos de corte diferentes para cada cromossomo pai. Em seguida, as subcadeias de cada cromossomo pai

[24] Do inglês, *Two Point Crossover*.
[25] Do inglês, *Mask Crossover*.

Figura 47 – Recombinação de dois pontos baseada em altura.



(a) Cromossomos Pais, c_1 e c_2. (b) Descendentes, c'_1 e c'_2.

Fonte: adaptado de Kaur, Chhabra e Singh (2010a) e Kaur, Chhabra e Singh (2010b).

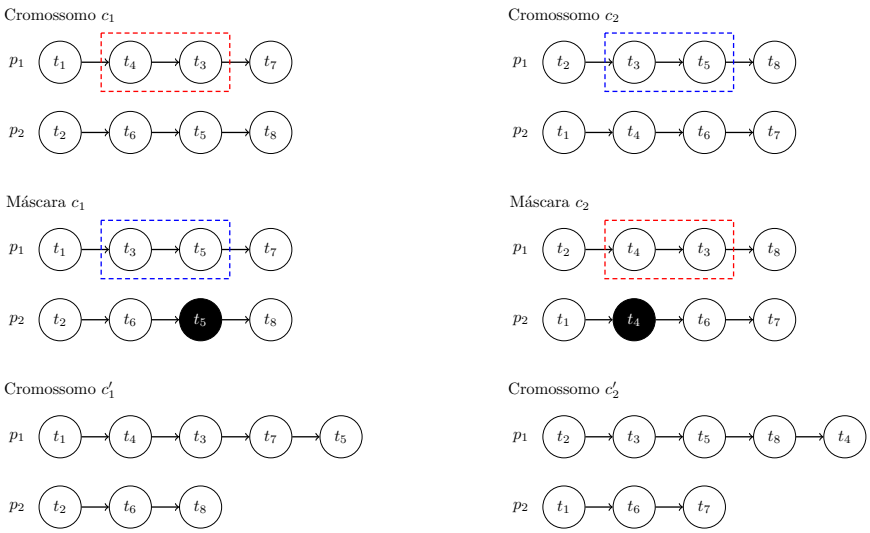
são trocadas. Os novos cromossomos formados a partir da troca das subcadeias são denominados cromossomos de máscara. Desta forma, uma máscara é formada para cada cromossomo que recebe uma nova subcadeia. Na próxima etapa, segundo a definição de Daoud e Kharma (2011), se as tarefas dispostas na subcadeia inserida não repetem na subcadeia removida, então as tarefas dispostas no cromossomo de máscara que repitam na subcadeia inserida são demarcados como NM (não mova). As Figuras 48(a) e 48(b) ilustram exemplos da demarcação para um par de cromossomos c_1 e c_2.

Após a demarcação das tarefas, os cromossomos descendentes são formados por meio da comparação, lista por lista, tarefa por tarefa, entre os cromossomos pais e suas respectivas máscaras. Desta forma, cada tarefa disposta no cromossomo descendente é formado por (DAOUD; KHARMA, 2011):

- Caso a tarefa no cromossomo pai seja idêntica à tarefa disposta no cromossomo de máscara, a tarefa é copiada ao cromossomo descendente;
  - Caso a tarefa esteja marcada como NM, então a tarefa é removida do cromossomo pai e do cromossomo máscara;
- Caso a tarefa no cromossomo pai seja diferente da tarefa disposta no cromossomo de máscara, somente a tarefa do cromossomo pai é copiada ao cromossomo descendente;
- Antes de avançar para a próxima lista, todas as tarefas dispostas no cromossomo de máscara, que não repetem na lista do cromossomo descendente, são copiadas na mesma ordem ao cromossomo descendente.

As Figuras 48(a) e 48(b) complementam o exemplo anterior ao apresentarem os cromossomo descendentes resultantes da recombinação.

Figura 48 – Recombinação de Máscara.



(a) Pai c_1 e filho c'_1. (b) Pai c_2 e filho c'_2.

Fonte: adaptado de Daoud e Kharma (2011).

4.3.1.3 Mutações

Nesta seção, são apresentadas diversas mutações utilizadas na TLE. Inicialmente, a primeira subseção apresenta mutações com deslocamento de tarefas. Em seguida, nas próximas subseções, são apresentados modelos de mutação baseados em definição de novos índices de altura, utilização do tempo de ociosidade, troca de tarefas, duplicação de tarefas, remoção de tarefas, ordenação por dígrafo, troca de processadores, reordenação de tarefas e definição de novas alocações.

Mutação de Troca por Altura (HSM)

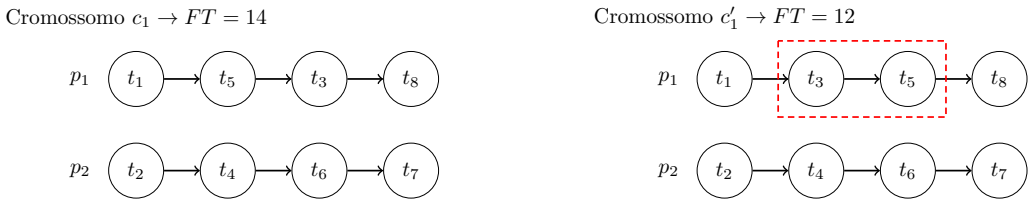
Inicialmente proposta em Hou, Ansari e Ren (1994), a mutação baseada em altura (HSM[26]) efetua uma troca de tarefas sem transgredir as restrições de precedência. Isso é possível uma vez que as tarefas trocadas possuem o mesmo valor de altura.

A Figura 49 apresenta um exemplo da HSM. O cromossomo c_1, disposto na Figura 49(a), sofre a mutação com troca das tarefas t_5 e t_6. Anteriormente, possuindo um tempo de finalização de 14, o novo cromossomo, c'_1, apresentado em 49(b), possui um tempo de finalização de 12. Portanto, o exemplo apresenta uma melhora significativa de performance no escalonamento com o uso da mutação.

Os passos aplicados na mutação baseada em altura estão sumarizados no Algoritmo 18. A geração de um novo cromossomo modificado é atribuída ao passo 3, uma vez que

[26] Do inglês, *Height Swap Mutation*.

Figura 49 – Mutação por Altura.



(a) Antes da Mutação. (b) Depois da Mutação.

Fonte: adaptado de Hou, Hong e Ansari (1990).

os passos anteriores são necessários para assegurar uma troca que obedeça às restrições de precedência.

Algoritmo 18 Mutação por Altura – adaptado de Hou, Ansari e Ren (1994)

entrada: cromossomos c_1
saída: cromossomos c'_1
{Seleção da tarefa}
1: Aleatoriamente, selecione uma tarefa, t_i
{Seleção de tarefa com altura equivalente}
2: Procure por uma tarefa, t_j, com a mesma altura.
{Troca de tarefas}
3: Construa um novo cromossomo, c'_1, através da troca das duas tarefas, t_i e t_j no escalonamento.

Jezic et al. (1999), Zomaya, Ward e Macey (1999), Singh e Pillai (2014) e Pillai et al. (2018) são outros exemplos de trabalhos que também aplicam o operador. Em contrapartida, Daoud e Kharma (2011) aplicam o operador sem se basear na altura das tarefas, porém, combinado a um mecanismo decodificador que garante a geração de soluções válidas. Kaur, Chhabra e Singh (2010a), Kaur, Chhabra e Singh (2010b) e Singh e Pillai (2014) apresentam uma versão desse operador onde a troca de tarefas pode ser estendida também à troca de processadores.

Mutação Interna de Altura (IHM)

Com o intuito de promover novas formas de exploração do espaço de busca, Tsujimura e Gen (1996) elaboraram três operadores, denominados 1, 2 e 3. A principal diferença dos operadores são os alvos de aplicação e a função altura utilizada, neste caso, o emprego da função $altura'$ – apresentada na Subseção 4.3.1.1. A Mutação Interna de Altura (IHM[27]) é representada pelos operadores 1 e 2 (IHM1 e IHM2), sendo estes mutações executadas entre os processadores de cada cromossomo em TLE, isto é, a operação reuni os aspectos do SPX (Subseção 4.3.1.2), todavia, aplicada internamente em cada cromossomo. Desse

[27] Do inglês, *Internal Height Mutation*.

modo, a relação intra-processador nessa mutação é a aplicação individual dos operadores. O operador 3 é similar à mutação de troca de tarefas, sendo abordado na Subseção 4.3.1.3.

O operador IHM1 trabalha com a definição de um ponto de corte, x, definido entre 1 e a $altura'$ máxima encontrada. O ponto de corte é posicionado em um intervalo restrito por duas regras: (i) as tarefas ao lado esquerdo devem possuir o valor de $altura'$ menor do que x; (ii) as tarefas do lado direito possuem valor de $altura'$ maior ou igual do que x. As restrições, quanto ao posicionamento do corte, garantem que a operação não construirá um cromossomo inválido. Em resumo, a execução do IHM1 está sumarizada no Algoritmo 19.

Algoritmo 19 Mutação Interna de Altura 1 – adaptado de Tsujimura e Gen (1996)

entrada: Cromossomo c_i;
saída: Cromossomo c'_i;
1: Gere um número aleatório, x, entre $[1, \max altura']$;
2: Acrescente o ponto de corte em cada processador de c_i de tal forma que a $altura'$ das tarefas antes do ponto de corte sejam menor que x e maior ou igual a x depois do ponto de corte. Desta forma, delimita-se duas metades dividas em cada processador;
3: Troque as metades dividas pelo ponto de corte.
4: **retorne:** c'_i

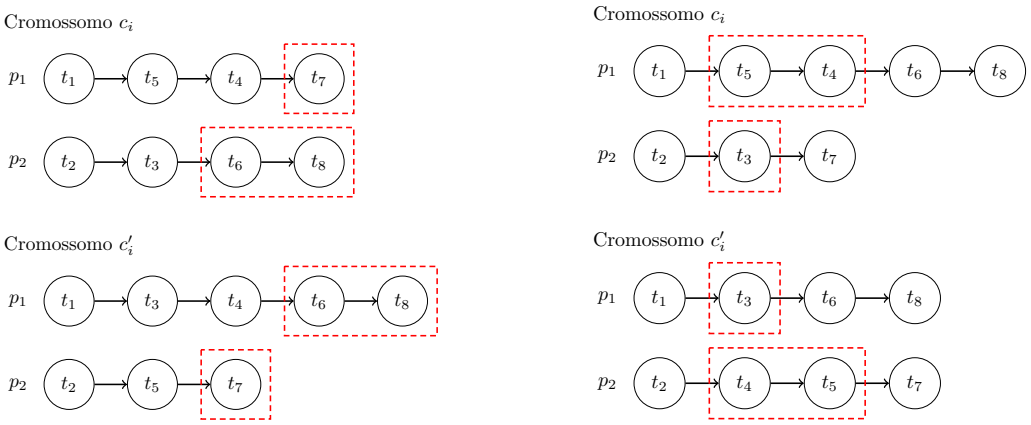
O operador IHM2 apresenta funcionamento análogo ao IHM1, exceto pela restrição empregada ao ponto de corte x. Neste caso, tarefas que possuem o valor de $altura' = x$ estarão no intervalo definido pelo ponto de corte. As instruções do Operador 2 estão enumeradas no Algoritmo 20.

Algoritmo 20 Mutação Interna de Altura 2 – adaptado de Tsujimura e Gen (1996)

entrada: Cromossomo c_i;
saída: Cromossomo c'_i;
1: Gere um número aleatório, x, entre $[1, \max altura']$;
2: Em cada processador de c_i, selecione todas as tarefas com $altura'$ igual a x;
3: Substitua a posição de todas as tarefas randomicamente.
4: **retorne:** c'_i

A Figura 50 apresenta exemplos de aplicação dos operadores IHM1 e IHM2. O operador IHM1, aplicado como exemplo da Figura 50(a), possui como valor de corte, $x = 2$. Portanto, supondo que $altura'(t_4) = 1$, a troca é efetuada entre os processadores de um mesmo cromossomo, uma vez que a restrição $altura'(t_6)$, $altura'(t_7)$ e $altura'(t_8) \geq 2$ foi atendida. Em contrapartida, o exemplo da Figura 50(b) aplica o operador IHM2. A operação é efetuada com um valor de $x = 1$, de forma que $altura'(t_4) = 1$. A restrição relacionada ao operador IHM2 também é atendida, já que $altura'(t_3)$, $altura'(t_4)$ e $altura'(t_5) = 1$.

Figura 50 – Mutações dos operadores IHM1 e IHM2.



(a) Operador IHM1. (b) Operador IHM2.

Fonte: adaptado de Tsujimura e Gen (1996).

Mutação de Alteração de Altura (HCM)

Três novos operadores foram criados em Tsujimura e Gen (1996). Dentre estes, os operadores de recombinação 1 e 2 foram abordados na Subseção 4.3.1.3. Neste contexto, o operador 3 é classificado como uma mutação, uma vez que a operação não apresenta permutações entre dois ou mais indivíduos.

Ao observar a geração da população inicial, Tsujimura e Gen (1996) constatou que o valor de $altura'$ não era alterado em qualquer fase do algoritmo. Em consequência disso, um terceiro operador foi criado para eliminar essa inconveniência e expandir a exploração do espaço de busca para encontrar um cromossomo melhor. O Algoritmo 21 sumariza as instruções do Operador 3, Mutação de Alteração de Altura (HCM[28]).

Algoritmo 21 Mutação de Alteração de Altura – adaptado de Tsujimura e Gen (1996)

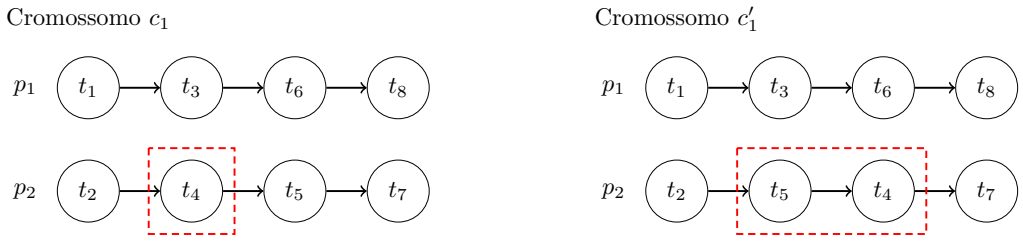
entrada: Cromossomo c_1;
saída: Cromossomo c'_1;
1: Selecione todas as tarefa de c_1 que $altura'$ pode ser alterada;
2: Selecione algumas dessas tarefas com uma pequena probabilidade;
3: Altere a $altura'$ das tarefas selecionadas em algum valor que as tarefas possam assumir. Mova as tarefas para uma posição apropriada de acordo com a $altura'$ alterada;
4: **retorne:** c'_1

A Figura 51 contempla um exemplo de aplicação da HCM sobre um cromossomo construído com o grafo da Figura 37. A alteração do valor de $altura'$ só pode ser efetuado em uma tarefa que apresente essa variação. No exemplo da Figura 51(a), somente a tarefa t_4 poderá ser alterada. Dessa forma, $altura'(t_4) = 1 \rightarrow 2$. Após a alteração, a

[28] Do inglês, *Height Change Mutation*.

restrição topológica impõe um conflito com a tarefa t_5, uma vez que $altura'(t_5) = 1$. Com as instruções finais da HCM, uma reordenação topológica. é efetuada para garantir a viabilidade do escalonador.

Figura 51 – Mutação de Alteração de Altura.



(a) Cromossomo válido. (b) Cromossomo alterado.

Fonte: adaptado de Tsujimura e Gen (1996).

Mutação de Troca por Ociosidade (ISM)

Esta mutação é aplicada quando o grafo de tarefas apresenta custo de comunicação. Proposta em Woo et al. (1997), a ideia é reduzir o tempo de ociosidade de um processador que espere por dados de outros processadores. A Expressão 24 define a tarefa, predecessora de uma tarefa t_i, com a maior quantidade de dados para distribuir (DDP[29]) à sucessora t_i. Esse operador é denominado Mutação de Troca por Ociosidade (ISM[30]):

$$DDP(t_i) = \{tk | \max_{t_k \in Pred(t_i)}(cc_{ki})\} \tag{24}$$

Utilizando a definição de DDP e as funções $extrair\_tarefas$ e $inserir\_tarefas$ (definidas e também utilizadas no TTX (Subseção 4.3.1.2)), o Algoritmo 22 define as instruções utilizadas na mutação baseada em ociosidade.

Algoritmo 22 Mutação de Troca por Ociosidade – adaptado de Woo et al. (1997)

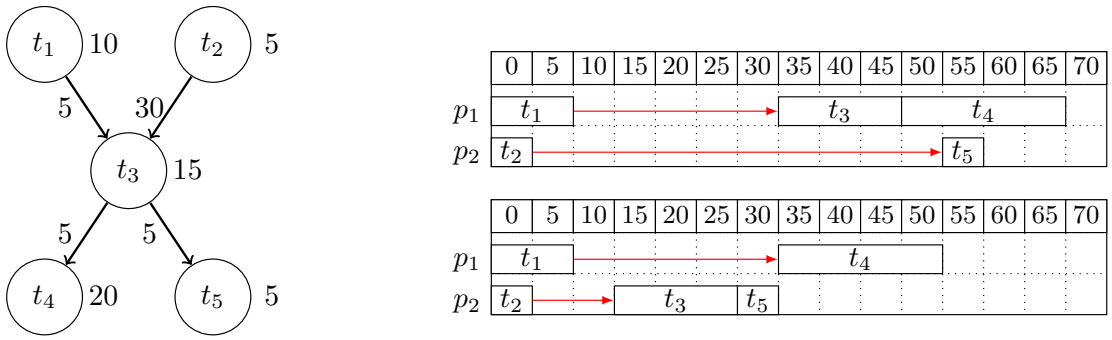
entrada: cromossomo c_1
saída: cromossomo c'_1

1: Selecione uma lista (processador) P_u, que tenha o maior tempo de finalização dentre os demais;
2: Na lista selecionada, encontre o maior período de ociosidade, também encontre a tarefa t_i disposta após o término desse período;
3: $extrair\_tarefas(X, p_u, t_i)$;
4: $inserir\_tarefas(X, p_v, t_i)$, onde p_v é o processador onde $DDP(t_i)$ está assinalado.

[29] Do inglês, *Data Dominating Parent.*
[30] Do inglês, *Idle Swap Mutation.*

A Figura 52 ilustra um exemplo da mutação baseada em ociosidade. A lista escolhida pelo Algoritmo 22 é p_1, uma vez que esta apresenta o maior tempo de finalização. Após identificar e remover a tarefa t_3, a função $inserir\_tarefas$ insere a tarefa t_3 na lista que contem a tarefa pai com a maior quantidade de dados para serem enviados, neste caso, a lista p_2.

Figura 52 – Mutação de Troca por Ociosidade.



(a) DAG com 5 tarefas. (b) Alteração no Gráfico de Gantt.

Fonte: adaptado de Woo et al. (1997).

Mutação por Troca (SM), Duplicação (DM) e Remoção (RM)

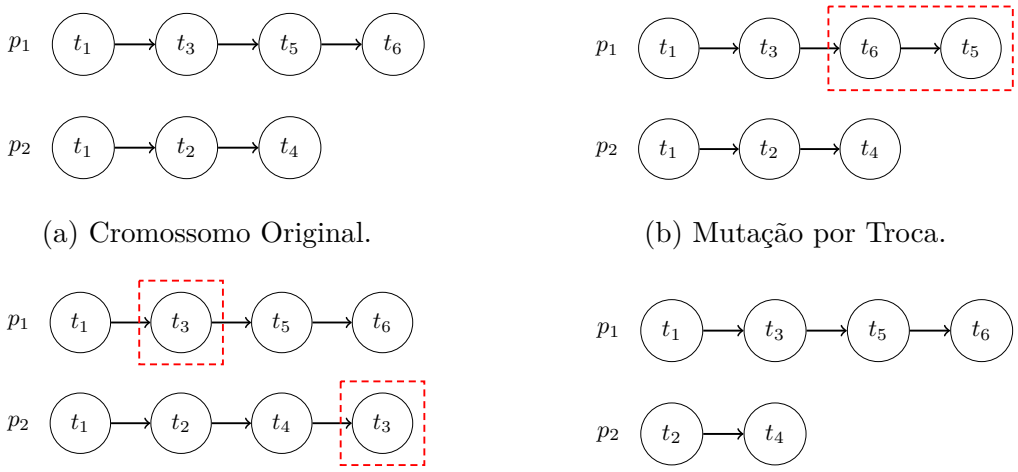
Utilizando a codificação de listas topológicas juntamente ao mecanismo de duplicação de tarefas, Tsuchiya, Osada e Kikuno (1997) consideraram aplicar três tipos de mutações em cromossomos que pudessem continuar satisfazendo as condições 1, 2 e 3 – apresentadas na Subseção 4.3.1.1. Esses operadores são a Mutação por Troca (SM[31]), por Duplicação (DM[32]) e por Remoção (RM[33]). Considerando $tarefa(i, p)$ como a i-ésima tarefa da lista do processador p e ao assumir que para uma $tarefa(i, p)$ uma mutação é executada, sendo uma dentre três operações, um operador é aleatoriamente executado (TSUCHIYA; OSADA; KIKUNO, 1997).

A Figura 53 ilustra as três mutações aplicadas individualmente no cromossomo disposto na Figura 53(a) e construído sobre o grafo da Figura 22(b). Inicialmente, a Figura 53(b) contempla uma troca entre as tarefa t_5 e t_6, em consequência da aplicação do operador à $tarefa(4, p_1)$, isto é, uma troca entre $tarefa(4, p_1)$ e $tarefa(3, p_1)$. A troca entre $tarefa(i, p)$ e $tarefa(i-1, p)$ só pode ser executada sobre as restrições $tarefa(i-1, p) \ll tarefa(i, p)$ e $i \geq 2$ (TSUCHIYA; OSADA; KIKUNO, 1998). Com o intuito de reduzir os custos de comunicação, o operador de duplicação de tarefas efetua a cópia de alguma tarefa para outra lista de processamento. O exemplo demonstrado na

[31] Do inglês, *Swap Mutation*.
[32] Do inglês, *Duplication Mutation*.
[33] Do inglês, *Removal Mutation*.

Figura 53(c) aplica o operador sobre $tarefa(2, p_1)$, consequentemente, copiando a tarefa t_3. Uma vez que não há qualquer tarefa sucessora de t_3 na lista alvo da cópia, a tarefa é inserida ao final da lista. Caso contrário, se alguma tarefa sucessora de t_3 estivesse presente, a tarefa seria inserida antes do primeiro sucessor da tarefa, de forma a satisfazer a condição 3. A duplicação só é executada se existir pelo menos uma lista em que a tarefa não esteja assinalada (TSUCHIYA; OSADA; KIKUNO, 1998). A mutação por remoção permite a remoção de alguma tarefa da lista e somente é executada quando a tarefa está assinalada em outra lista (TSUCHIYA; OSADA; KIKUNO, 1998). A Figura 53(d) ilustra a remoção da tarefa t_1 a partir da execução do operador sobre $tarefa(1, p_2)$ (TSUCHIYA; OSADA; KIKUNO, 1997).

Figura 53 – Mutação por Troca, Duplicação e Remoção.



(a) Cromossomo Original.

(b) Mutação por Troca.

(c) Mutação por Duplicação.

(d) Mutação por Remoção.

Fonte: adaptado de Tsuchiya, Osada e Kikuno (1997).

Singh e Singh (2012) é um outro exemplo de trabalho que aplica a SM.

Mutação por Reordenação de Dígrafo (DRM)

Proposta por Corrêa, Ferreira e Rebreyend (1999), a Mutação por Reordenação de Dígrafo (DRM[34]) está associada à TLE ordenada por dígrafos (Subseção 4.3.1.1). O primeiro passo é construir um grafo $D(c) = (T, E(c))$, onde c é o cromossomo a ser modificado. Posteriormente, o novo indivíduo será gerado através do método utilizado para gerar a população inicial (heurística de lista) combinado à politica de ordenação. A heurística de lista é formada por um processo interativo de escalonamento, onde cada tarefa torna-se livre após o escalonamento de todas as suas respectivas tarefas precedentes. Tarefas livres são designadas aos processadores aleatoriamente. Nesse operador, o conjunto F de tarefas livres é determinado de acordo com $D(c)$ (CORRÊA; FERREIRA; REBREYEND, 1999).

[34] Do inglês, *Digraph Reorder Mutation*.

Corrêa, Ferreira e Rebreyend (1999) também implementam uma versão desse operador onde, dado um cromossomo c, após a geração do dígrafo $D(c) = (\tau, E(c))$, o novo cromossomo c' é formado através de heurísticas, sendo o mesmo algoritmo utilizado na segunda versão da recombinação baseada em dígrafo (Subseção 4.3.1.2). Entretanto, com uma pequena modificação na regra 1: o tempo de introdução mínimo das tarefas é calculado exclusivamente sobre as restrições de precedência.

Mutação por Troca de Processadores (PSM)

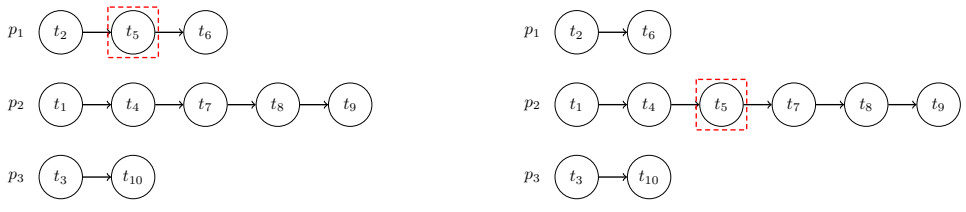
Além de proporem uma recombinação (Subseção 4.3.1.2), Golub e Kasapovic (2002) também apresentaram um operador de Mutação por Troca de Processadores (PSM[35]). Dois números inteiros não iguais são sorteados para atuarem como índices ligados às listas de processadores. O Algoritmo 23 sumariza as instruções do operador destacando as duas principais restrições da aplicação.

Algoritmo 23 Mutação por Troca de Processadores – adaptado de Golub e Kasapovic (2002)

entrada: Cromossomo c_1;
saída: Cromossomo c'_1
1: Gere dois números, aleatoriamente, r e $q \in [1, P]$ com as condições:
2: a) $r \neq q$;
3: b) conjunto r não pode estar vazio.
4: Em c_1, no conjunto r, aleatoriamente, retire uma tarefa de e a coloque no conjunto q.
5: **retorne:** c'_1

Um exemplo onde $r = 1$ e $q = 2$ é apresentado na Figura 54. Na simulação, a tarefa t_5 é removida do processador 1 e realocada ao processador 2. De acordo com as restrições de precedência, a tarefa t_5 é posicionada topologicamente.

Figura 54 – Mutação por Troca de Processadores.



(a) Cromossomo. (b) Aplicação da Mutação.

Fonte: adaptado de Golub e Kasapovic (2002).

Kaur, Chhabra e Singh (2010a) e Kaur, Chhabra e Singh (2010b) são outros exemplos de trabalhos que aplicam o operador.

[35] Do inglês, *Processor Swap Mutation*.

Mutação de Reordenação (SRM)

A Mutação por Reordenação (SRM[36]), proposta inicialmente por Zhong e Yang (2003), utiliza um algoritmo auxiliar, denominado MakeSeqList, para gerar uma nova lista de sequências. Posteriormente, a lista é utilizada como parâmetro para a reordenação das tarefas de um escalonamento s. A aplicação da mutação está simplificada no Algoritmo 24.

Algoritmo 24 Mutação de Reordenação – adaptado de Zhong e Yang (2003)

entrada: Cromossomo c_1
saída: Cromossomo c_1'
1: Execute o algoritmo MakeSeqList para gerar uma lista de sequência SL
2: **para todo** p_i em P **faça**
3: Reordene a ordem de execução das tarefas em p_i de acordo com a relação de precedência em SL.
4: **fim para**

Com a aplicação do MakeSeqList, (Algoritmo 25), Zhong e Yang (2003) garantem que todos os escalonadores factíveis podem ser alcançados com alguma probabilidade, uma vez que o algoritmo é baseado exclusivamente no DAG e sua construção deve ser efetuada em cada iteração do GA, precisamente, na fase de mutação.

Algoritmo 25 Algoritmo MakeSeqList – adaptado de Zhong e Yang (2003)

entrada: Conjunto de tarefas T;
saída: Lista $SeqList$;
1: Lista $SeqList \leftarrow \{\}$;
2: Lista $F \leftarrow \{t_1\}$;
3: Lista $R \leftarrow V - \{t_1\}$;
4: **enquanto** $F \neq \{\}$ **faça**
5: Escolha aleatoriamente uma tarefa t_i e adicione ao final de $SeqList$
6: Mova cada tarefa t_j em R para F se todos os predecessores de t_j estiverem em $SeqList$
7: $F \leftarrow F - t_i$
8: **fim enquanto**

Mutação de Processador (PM)

O operador de Mutação de Processador (PM[37]) aleatório também efetua uma troca de alocação. Entretanto, diferentemente da PSM (Subseção 4.3.1.3), o operador define um novo processador para uma única tarefa selecionada. Portanto, ao contrário de trocar alocações já definidas entre tarefas selecionadas, uma única alocação é substituída. Este

[36] Do inglês, *Sort Mutation*.
[37] Do inglês, *Processor Mutation*.

operador é aplicado nos trabalho de Kaur, Chhabra e Singh (2010a) e Kaur, Chhabra e Singh (2010b).

Singh (2012) apresenta uma variação deste operador que exclui a aleatoriedade na escolha do novo processador. Nessa adaptação, o novo processador escolhido é definido como aquele que apresenta o tempo de disponibilidade mínimo dentre todos os processadores disponíveis. Ao observarem que, dependendo dos operadores, cromossomos não modificavam sua disposição de tarefas, Zomaya, Ward e Macey (1999) apresentam outra variação do operador ao forçarem a troca do processador pra realizar mudanças nas listas no decorrer das gerações.

4.3.2 Codificação por Matrizes (ME)

Codificar soluções em matrizes é uma possível abordagem para encontrar soluções no MTSP. A Codificação por Matrizes (ME[38]) é formada pela junção de dois arranjos bidimensionais, as matrizes de escalonamento e alocação, ambas de dimensão $n \times n$, onde n é a quantidade total de tarefas dispostas no grafo. O emprego das matrizes fornece uma visão clara sobre o espaço de busca com informações adicionais das restrições e das alocações efetuadas. Na codificação por matrizes para o MTSP, a aptidão dos indivíduos é extraída com base no tempo final de execução do escalonamento. A representação descrita nesta seção é embasada no trabalho de Wang e Korfhage (1995), que fornece definições e estruturas da codificação de matrizes em um ambiente com processadores homogêneos.

As principais funções utilizadas na codificação proposta por Wang e Korfhage (1995) estão resumidas a seguir:

- $descendente(t_i)$: mostra todas as tarefas alcançáveis a partir de t_i;
- $antecessor(t_i) = \{t_j | t_i \in descendente(t_j)\}$: mostra tarefas que são possíveis antecedentes de t_i;
- $pred(t_i) = \{t_j | (t_j, t_i) \in E\}$: mostra o conjunto de tarefas predecessoras imediatas de t_j;
- $prev(R, t_j) = t_i$: mostra a tarefa t_i que precede t_j no mesmo processador.

Ao propor essa codificação, Wang e Korfhage (1995) dividem a estruturação de matrizes em três definições, sendo que a primeira aborda a matriz de escalonamento e as demais lidam com a matriz de alocação. As definições são:

[38] Do inglês, *Matrix Encoding*.

- **Definição 1:** uma matriz de escalonamento, MS, é uma matriz $n \times n$, onde o valor de cada célula MS_{ij} é determinado por:

$$MS_{ij} = \begin{cases} 1 & \text{, se } t_i \notin desc(t_j) \text{ e } i \neq j \\ 0 & \text{, caso contrário} \end{cases} \tag{25}$$

  A matriz de escalonamento contempla as restrições de precedências impostas pelo grafo de tarefas. Nesse contexto, as células assinaladas com 1 na coluna j identificam quais tarefas podem ser escalonadas antes da tarefa t_j. Em contrapartida, as células assinaladas com 0 na coluna j identificam as tarefas que não devem ser assinaladas antes da tarefa t_j. Os operadores de recombinação e mutação são aplicados exclusivamente na matriz de alocação, uma vez que a matriz de escalonamento torna-se imutável ao identificar as variadas possibilidades de execução das tarefas, isto é, não necessariamente uma configuração de execução em ordem específica. Um exemplo de matriz de escalonamento é apresentado na Figura 55(b), construído sob o grafo de tarefas da Figura 55(a).

- **Definição 2:** uma matriz de alocação, MA, é uma matriz $n \times n$, na qual o valor de cada célula MA_{ij} é determinado por:

$$MA_{ij} = \begin{cases} 1 & \text{, se } prev(R, t_j) = t_i, \\ 0 & \text{, caso contrário} \end{cases} \tag{26}$$

  Uma matriz de alocação carrega a codificação necessária para identificar a ordem de execução das tarefas atribuídas a cada um dos processadores. Ao identificar a tarefa t_i designada antes de t_j, em um mesmo processador, dado um escalonamento R, a função $prev(R, t_j)$ permite a construção da sequência de execução de cada processador. Além disso, é necessário definir uma função adicional voltada a identificar incoerências advindas de precedências mal sincronizadas. Dessa forma, a função $Conjunto\_Antecessor(MA, t_i)$ que corrobora com a terceira definição, pode ser definida como:

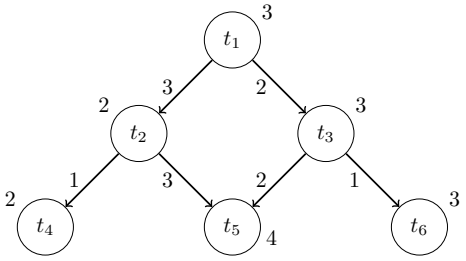
$$\begin{aligned} Conjunto\_Antecessor(MA, t_i) = \{t_j \cup Ancestor(MA, t_j)\} \\ \text{para todo } t_j \in \{Pred(t_i) \cup Prev(t_i)\} \end{aligned} \tag{27}$$

- **Definição 3:** uma matriz de alocação, MA, será exequível se $t_i \notin Conjunto\_Antecessor(MA, t_i)$ para todas tarefas t_i em MA.

A Figura 56 apresenta exemplos de matrizes de alocação, onde cada matriz apresenta seu respectivo plano de alocação. As matrizes são embasadas na Figura 55. O escalonamento gerado pela matriz disposta na Figura 56(a) não apresenta problemas de

sincronização, isto é, todas as tarefas atendem a terceira definição. Todavia, o escalonamento advindo da Figura 56(b) é considerado inválido. A alocação assinalada na célula MA_{16} produz incoerências, algumas das quais podem ser observadas no seu respectivo escalonamento. Por exemplo, a alocação inicial das tarefas t_6 e t_2 gera um impasse nas condições de precedências impostas pelo grafo de tarefas.

Figura 55 – Matriz de escalonamento na ME.



(a) DAG com 8 tarefas.

	1	2	3	4	5	6
1	0	1	1	1	1	1
2	0	0	1	1	1	1
3	0	1	0	1	1	1
4	0	0	1	0	1	1
5	0	0	0	1	0	1
6	0	1	0	1	1	0

(b) Matriz de escalonamento.

Fonte: do autor.

Figura 56 – Matriz de alocação na ME.

	1	2	3	4	5	6
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	1
6	0	0	0	0	0	0

$p_1 : 1 \rightarrow 3 \rightarrow 4$
$p_2 : 2 \rightarrow 5 \rightarrow 6$
(a) MA válida.

	1	2	3	4	5	6
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	1	0	0	0	0	0

$p_1 : 2 \rightarrow 5$
$p_2 : 6 \rightarrow 1 \rightarrow 3 \rightarrow 4$
(b) MA inválida.

Fonte: do autor.

Com o objetivo de reduzir o espaço de busca a somente nós executáveis, Wang e Korfhage (1995) definem três propriedades dos cromossomos baseados na codificação por matrizes. Aplicadas essencialmente em matrizes de alocação, essas propriedades são:

1. $MA_{ij} \cap MS_{ij} = MA_{ij}$, onde MS é uma matriz escalonável corresponde de MA.

A primeira propriedade aponta a coerência entre as matrizes de alocação e escalonamento. Dessa forma, as células que definem as alocações em uma matriz devem respeitar as restrições de ordem indicadas pela matriz de escalonamento.

2. Se $MA_{ij} = 1$, então $MA_{ik} = MA_{ik} = MA_{lj} = 0$ para todo $k, l = 1...n$ e $k \neq j$ e $l \neq i$.

   A segunda propriedade identifica a quantidade de uns disposta em uma coluna juntamente a linha correspondente. Dessa forma, se uma determinada célula MA_{ij} assinalar o valor 1, as demais células da coluna e linha devem ser preenchidas com o valor zero. Uma linha que apresente mais de uma célula com o valor 1 identificaria que uma tarefa t_i está alocada em mais de um processador, sendo semelhante a duplicação de tarefas, todavia, não abordada neste modelo. Ademais, uma coluna que apresente mais de uma célula com o valor 1 invalida a função $prev(R, t_j)$, uma vez que não seria possível identificar explicitamente qual tarefa t_i antecede t_j.

3. $n - m <= \sum_{i=1}^{n} \sum_{j=1}^{n} MA_{ij} <= n - 1$.

   A terceira propriedade traça limites nas alocações efetuadas. A primeira restrição $(n - m)$ delimita a quantidade mínima de alocações considerando a quantidade de tarefas e processadores. A restrição final $(n - 1)$ restringe a quantidade máxima de alocações possíveis, considerando que pelo menos uma tarefa não possua uma tarefa antecedente.

4.3.2.1 Recombinação de Ponto Único (SPX)

Na codificação por matrizes, o SPX é adotado como operador genético responsável por combinar os genes dos cromossomos. Nesse modelo, após a seleção de dois indivíduos, define-se aleatoriamente um ponto de corte entre 1 e $n - 1$. Em seguida, as matrizes de alocação de cada indivíduo são divididas através do ponto de corte, posteriormente, ocorrendo a troca e combinação das metades à direita do corte. Este processo é ilustrado pela Figura 57.

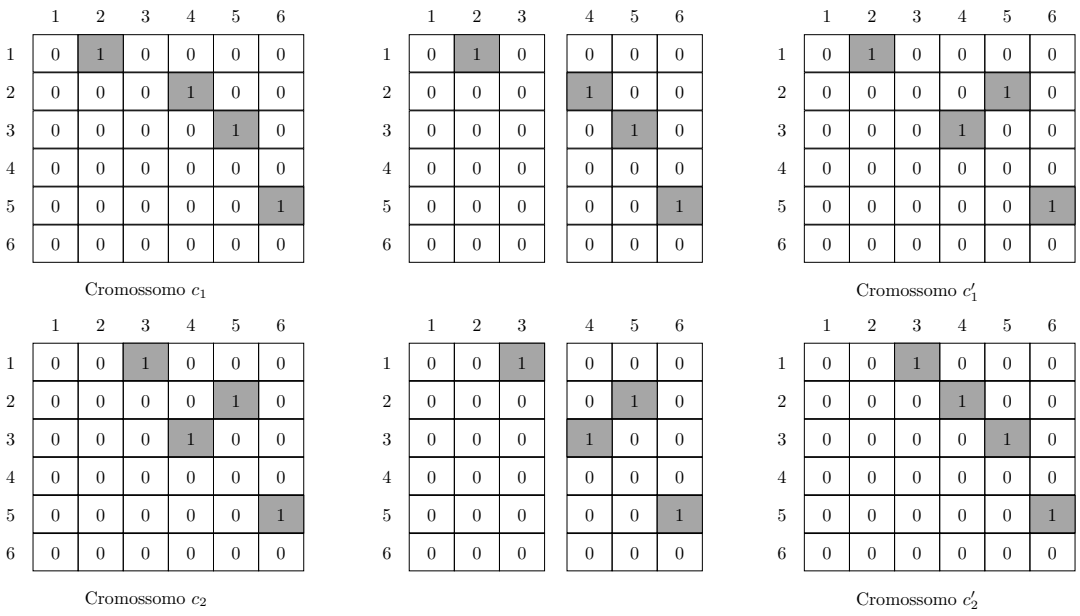
Apesar da simplicidade na recombinação de um ponto, indivíduos não executáveis podem ser gerados após a operação. Em um cenário com espaço de busca limitados ao indivíduos válidos, Wang e Korfhage (1995) apresentam três operadores voltados à adaptação de soluções inválidas, sendo esses[39]:

- ❑ **Removedor de duplicação:** aplicado quando existe mais de uma célula com valor 1 em uma linha, onde é efetuada a remoção da quantidade extra de uns;

- ❑ **Reparador de viabilidade:** aplica-se o operador se existir uma dada tarefa t_i tal que $t_i \in Conjunto\_Antecessor(MA, t_i)$. A operação resume-se em selecionar

[39] No original denominados *repair-dup*, *make-realizable* e *repair-less*, respectivamente.

Figura 57 – Recombinação de Ponto Único na ME.



Fonte: do autor.

aleatoriamente t_k e $t_l \in Conjunto\_Antecessor(MA, t_i)$, onde $t_k = Prev(t_l)$ e $t_k \notin Descendente(t_l)$. Ao final, a correção é efetuada na atribuição $A_{kl} = 0$, de forma a eliminar a inconsistência;

- ❑ **Reparador de insuficiência:** quando a quantidade de alocações (uns) é menor que $n - m - 1$, é efetuada uma realocação baseada no primeiro processador com tempo disponível e consequentemente, convertendo a matriz para uma que não use mais do que uma quantidade m de processadores.

4.3.2.2 Mutação por Complementação (CM)

O operador de mutação utilizado na codificação de matrizes é caracterizado pela complementação de valores em posições aleatórias da matrizes de alocação. Assim como a recombinação, a Mutação por Complementação (CM[40]) também pode gerar codificações não executáveis. Dessa forma, os reparadores de viabilidade e insuficiência citados anteriormente são aplicados para a obtenção de codificações válidas (WANG; KORFHAGE, 1995).

[40] Do inglês, *Complementation Mutation*.

4.3.3 Codificação de Alocação e Escalonamento (MSE)

A codificação de alocação e escalonamento (MSE[41]) é o modelo mais adotado na literatura analisada neste trabalho. Apresentada por Wang et al. (1997), a MSE controla a alocação das tarefas sobre processadores e a ordenação da sequência de execução. Desse modo, seguindo a definição de Wang et al. (1997), MSE foi proposta como uma representação que une as cadeia de alocação e ordenação, respectivamente *mat* e *ss*. Além disso, a definição da ordem de tarefas em cada processador e o escalonamento dos dados globais de transmissão são deixados para a etapa de avaliação do cromossomo.

Definindo o processador que executará uma tarefa, *mat* é um cadeia de tamanho n, de forma que $mat(i) = p_j$, onde $0 \leq i < n$ e $0 \leq j < m$ (WANG; SIEGEL; ROYCHOWDHURY, 1996). De forma parecida, representando uma ordem de execução que respeite as restrições de precedência, a cadeia *ss* possui tamanho n, tal que $ss(k) = t_i$, onde $0 \leq i, k < n$, e cada tarefa t_i aparece uma única vez na cadeia (singularidade) (WANG; SIEGEL; ROYCHOWDHURY, 1996). O custo de comunicação entre as tarefas é relacionado à ordem de precedência, por exemplo, se um gene $ss(k)$ necessita de dados providos por uma tarefa no gene $ss(j)$, logo $j < k$. A concorrência da ordem de execução para tarefas alocadas a um mesmo processador é traduzida como a ordem de incidência das tarefas na cadeia *ss*, isto é, a tarefa disposta mais próxima do início de cadeia é a tarefa que executa primeiro na unidade.

Baseando no grafo de tarefas da Figura 22(a), as Figuras 58(a) e 58(b) ilustram a representação adotada em Wang, Siegel e Roychowdhury (1996), representando respectivamente as cadeias *ss* e *mat*. A Figura 58(c) apresenta as codificações adotadas em uma única abstração, de forma análoga aos trabalhos de Omara e Arafa (2009), Chitra et al. (2010), Gupta, Kumar e Agarwal (2010), Chitra, Venkatesh e Rajaram (2011), Mohamed e Awadalla (2011) , Sathappan et al. (2011), Kaur e Singh (2012), Panwar, Lal e Singh (2012), Awadall, Ahmad e Al-Busaidi (2013), Dhingra, Gupta e Biswas (2014) e Morady e Dal (2016).

Lee e Chen (2003) também utilizam MSE com algumas modificações, definindo uma pré-ordenação para as tarefas com base em partições formadas pelos valores de *b-level*. Essa estratégia incorpora o conceito de dividir para conquistar e melhora o desempenho do GA ao resolver as partições individualmente, combinando-as ao final. Conforme a definição de Lee e Chen (2003), o particionamento é formado em três etapas:

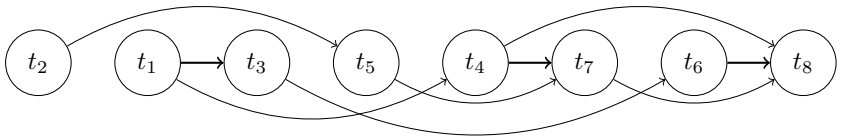
1. Calcular o *b-level* de cada tarefa;

2. Ordenar as tarefas em ordem decrescente de *b-level* (desempate é efetuado aleatoriamente);

3. Particione as tarefas em subgrupos uniformemente em sequência.

[41] Do inglês, *Matching Scheduling Encoding*.

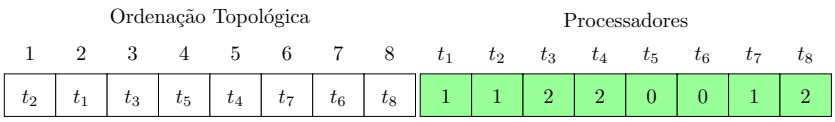
Figura 58 – Codificação de Alocação e Escalonamento.



(a) Cadeia de escalonamento.

$t_1 : 1 \quad t_2 : 1 \quad t_3 : 2 \quad t_4 : 2 \quad t_5 : 0 \quad t_6 : 0 \quad t_7 : 1 \quad t_8 : 2$

(b) Cadeia de alocação.

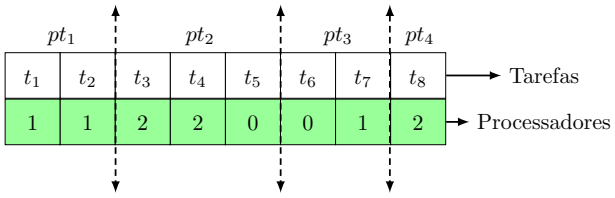


(c) Unindo alocação e escalonamento.

Fonte: adaptado de Wang, Siegel e Roychowdhury (1996).

Baseando-se no grafo da Figura 22(a), a Figura 59 apresenta um exemplo da MSE aplicada no trabalho de Lee e Chen (2003). Cada partição é formada de modo que as tarefas dispostas em uma partição não sejam necessárias à execução das tarefas nas partições anteriores, isto é, de forma que as partições não formem um ciclo de dependências no grafo de tarefas.

Figura 59 – MSE com partições baseadas em índice *b-level*.



Fonte: adaptado de Lee e Chen (2003).

O trabalho de Bonyadi e Moghaddam (2009) é um outro exemplo que aplica MSE com algumas modificações. Os autores dividem as cadeias de alocação e escalonamento, tratando cada cadeia como um único cromossomo. Desse modo, trabalhando um GA bipartido, as cadeias de alocação e escalonamento são processadas em GAs distintos, cada um com mecanismos exclusivos para lidar com cada tipo de população. Por exemplo, o GA que manipula as cadeias de escalonamento possui recombinações e mutações mais restritivas devido às restrições de precedência. Paralelamente, o GA voltado às cadeias de

alocação utiliza operadores mais simplificados, uma vez que os cromossomos não possuem tantas limitações. Entretanto, apesar da divisão, o valor de aptidão de cada cromossomo é calculado utilizando a união das cadeias de alocação ou escalonamento.

O estudo de Kang, Zhang e Chen (2011) apresenta a MSE em conjunto à heurística HELF (TOPCUOGLU; SEVILMIS, 2002) para estabelecimento das restrições de precedência. Além disso, os tempos de execução e de comunicação das tarefas são avaliadas por uma medida de atividade. Kang, Zhang e Chen (2011) destacam que a medida de atividade de um gene é empregada na tentativa de inferir as mudanças de desempenho na execução de uma tarefa. A Expressão 28 apresenta o cálculo de atividade de uma tarefa t_i, onde w_{ij} é o tempo estimado para concluir um tarefa t_i em um processador p_j.

$$ac_i = 1/10 * (w_{i,j} - w_i)/w_{i,j} \tag{28}$$

Ahmad, Munir e Nisar (2012) apresentam uma comparação entre dois GAs que utilizam a MSE. No primeiro GA, a MSE é empregada conforme o padrão ilustrado nos exemplos supracitados. Em contrapartida, o segundo algoritmo emprega a heurística de *b-level* para definir a ordenação das tarefas na cadeia de escalonamento. Dessa maneira, após a definição de classificações de prioridade (*ranks*), a cadeia de escalonamento se mantém inalterada após a aplicação dos operadores. A Figura 60 apresenta um exemplo, baseado no grafo da Figura 22, da utilização do *b-level* diretamente na MSE, onde a sequência gerada é formada pelas tarefas $t_1, t_2, t_4, t_5, t_3, t_6, t_7$ e t_8. Esse mesmo padrão de organização também é adotado em Ahmad et al. (2016).

Figura 60 – MSE com ordenação baseada em *b-level*.

Ordenação Topológica (b-level)								Processadores							
1	2	3	4	5	6	7	8	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
20	20	12	14	13	9	7	4	1	1	2	2	0	0	1	2

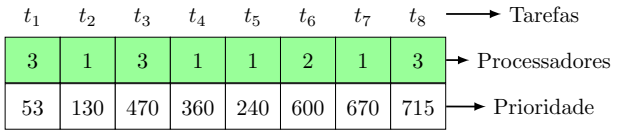
Fonte: adaptado de Ahmad, Munir e Nisar (2012).

Yao, You e Li (2004) implementam a TD por meio de algumas modificações na MSE. A primeira diferença está presente no cadeia de alocação, onde os genes não correspondem a um único processador, mas sim a uma possível coleção de processadores. Dessa forma, um gene da cadeia de alocação é composto por uma dentre 2^{i-1} combinações, para $i = 1...m$, onde m é a quantidade de processadores. Em contrapartida, a cadeia de escalonamento tem suas tarefas ordenadas baseando-se na altura do grafo de tarefas e é construída utilizando um valor de prioridade que respeite as restrições de precedência. Sendo a $prioridade(t)$ de uma tarefa t um valor entre 1 e MAX_MUM, a Expressão 29 determina os valores de prioridade normalizados (pri) para cada tarefa. Yao, You e Li (2004) destacam que quanto menor for o valor de pri, maior será a prioridade da

tarefa. Além disso, caso duas tarefas apresentem o mesmo valor de prioridade, a tarefa que efetuou o cálculo antes é priorizada. Baseando-se no grafo da Figura 22(a) em um cenário com dois processadores, a Figura 61 apresenta um exemplo das modificações empregadas por Yao, You e Li (2004) na MSE. Neste exemplo, a tarefa t_3 possui o valor 3 como código de alocação, sendo assim, t_3 está escalonada para p_1 e p_2. Uma vez que $pri(t3)(= 523) > pri(t_4)(= 413)$, t_3 começa após a conclusão de t_4 em p_1.

$$\begin{cases} pri(t_j) = prioridade(t_j) & \text{se } Pred(t_j) = \emptyset \\ pri(t_j) = \max_i pri(t_i) | t_i \in Pred(t_j) + prioridade(t_j) & \text{caso contrário} \end{cases} \tag{29}$$

Figura 61 – MSE baseada em prioridade e adaptada à duplicação de tarefas.



Fonte: adaptado de Yao, You e Li (2004).

4.3.3.1 Recombinações

Sendo uma das representações mais populares, a MSE possui diversas recombinações, sendo essas apresentadas nas próximas subseções. A primeira subseção apresenta uma versão do SPX. Uma adaptação dessa recombinação é apresentada na subseção seguinte. As demais subseções contextualizam recombinações com o uso de trocas dinâmicas, sequência principal, pesos com mapeamentos, emprego de dois pontos de corte, análise de atividade dos genes e outras formas de permutação.

Recombinação de Ponto Único (SPX)

No trabalho de Wang, Siegel e Roychowdhury (1996) e Wang et al. (1997), o operador SPX é aplicado nas diferentes cadeias da codificação. Inicialmente, na parte das alocações, ao selecionar aleatoriamente duas cadeias de alocação, um ponto de corte, também definido aleatoriamente, divide as cadeias em dois segmentos. Em seguida, os segmentos inferiores definidos pelo corte são trocados entre as cadeias, de forma que as novas cadeias geradas são formadas pela interseção de suas antigas e novas alocações. Com um GA específico para a cadeia de alocação, Bonyadi e Moghaddam (2009) também aplicam esse operador.

A recombinação de um ponto também é aplicada a um par de cadeias de escalonamento. A diferença quanto à aplicação nas cadeias de alocação está na forma de troca

dos segmentos inferiores. Atendendo as restrições de precedência, a troca dos segmentos inferiores é seguida de uma reordenação das tarefas. A nova ordem de cada segmento é baseada na posição relativa e identidade de cada tarefa no seu segmento de origem. Esse procedimento garante a não geração de cadeias de escalonamento inválidas.

O SPX implementado por Omara e Arafa (2009) trabalha com as recombinações de um ponto e de ordem. Ao selecionar um par de indivíduos, um número aleatório define qual tipo de recombinação será aplicada. Consequentemente, as diferentes recombinações são aplicadas em partes distintas do cromossomo. Por exemplo, caso selecionado, o SPX baseado em ordem é aplicado a cadeia de escalonamento. De maneira equivalente, o SPX é aplicado somente na cadeia de alocação.

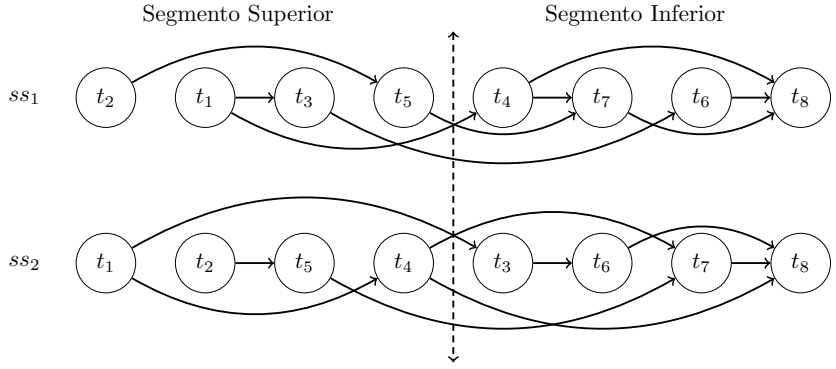
Baseando-se no modelo de Wang, Siegel e Roychowdhury (1996), a Figura 62(a) apresenta um exemplo da aplicação da recombinação de um ponto em um par de cadeias de escalonamento. Após o estabelecimento do ponto de corte, os segmentos são definidos, onde os segmentos inferiores são trocados. Após a troca, conforme a Figura 62(b), uma reordenação é aplicada para garantir as restrições de precedência. Por fim, duas novas cadeias de escalonamentos são formadas.

Análogo ao trabalho de Mohamed e Awadalla (2011), Awadall, Ahmad e Al-Busaidi (2013) empregam a MSE dentre as diferentes codificações apresentadas no seu trabalho. Entretanto, a cadeia de escalonamento de Awadall, Ahmad e Al-Busaidi (2013) emprega uma heurística min-min (SINGH; SINGH, 2012) para ordenação das tarefas.

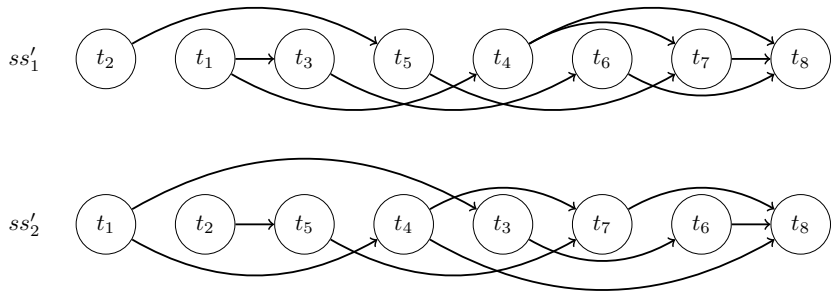
SPX também está presente nos operadores utilizado por Yao, You e Li (2004), Chitra et al. (2010), Chitra, Venkatesh e Rajaram (2011), Sathappan et al. (2011), Kaur e Singh (2012), Dhingra, Gupta e Biswas (2014) e Ahmad et al. (2016).

Mohamed e Awadalla (2011) e Awadall, Ahmad e Al-Busaidi (2013) aplicam a recombinação de um ponto nas três codificações adotadas nos trabalhos, sendo essas TOE, PLE e MSE. A aplicação na TOE respeita as restrições de precedência ao reordenar tarefas após o cruzamento, e a PLE utiliza o operador sem maiores restrições. Entretanto, uma vez que a MSE é uma combinação dessas codificações anteriores, Mohamed e Awadalla (2011) dividem a aplicação da recombinação em intervalos de aplicação, de forma que, dada uma probabilidade de cruzamento pc, se $0.35 \leq pc \leq 0.55$ o operador é aplicado na primeira parte (escalonamento), se $0.55 \leq pc \leq 0.75$ o operador é aplicado na segunda parte (alocação), e se $0.75 < pc \leq 0.95$ o operador é aplicado em ambas as partes. No primeiro GA proposto, Ahmad, Munir e Nisar (2012) também definem a cadeia alvo da aplicação do operador baseando-se em um valor aleatório em comparação à uma constante. Porém, em seu segundo GA, Ahmad, Munir e Nisar (2012) aplicam o SPX exclusivamente na cadeia de alocação, sendo seguido pela aplicação da recombinação de dois pontos.

Figura 62 – Recombinação de Ponto Único na MSE.



(a) Cadeias selecionadas.



(b) Resultado da operação.

Fonte: adaptado de Wang, Siegel e Roychowdhury (1996) e Wang et al. (1997).

Recombinação de Ponto Único com Partições (SBX)

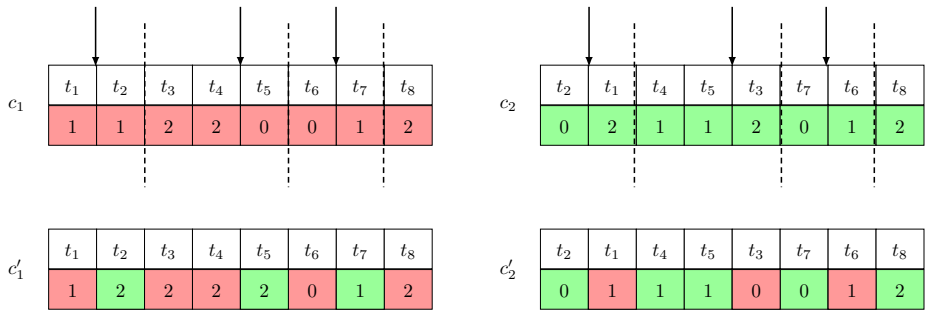
Reforçando seu modelo de divisão de tarefas, Lee e Chen (2003) também implementam o SBX na cadeia de alocação. De forma análoga, dados dois cromossomos, c_1 e c_2, seus descendentes, c'_1 e c'_2 recebem diretamente a sequência de execução definida pelos seus respectivos pais. A troca de segmentos é realizada somente entre as cadeias de alocação de c_1 e c_2, sendo o operador aplicado em cada partição do modelo. A Figura 63 ilustra um exemplo da aplicação da Recombinação de Ponto Único com Partições (PSPX[42]).

Recombinação de Pontos Aleatórios (RPX)

O segundo modelo de recombinação adotado por Lee e Chen (2003) também mantém inalteradas as cadeias de escalonamento dos pares selecionados. Essa característica elimina a necessidade de elaborar algoritmos de checagem e reparo, o que pode ser muito

[42] Do inglês, *Partition Single Point Crossover*.

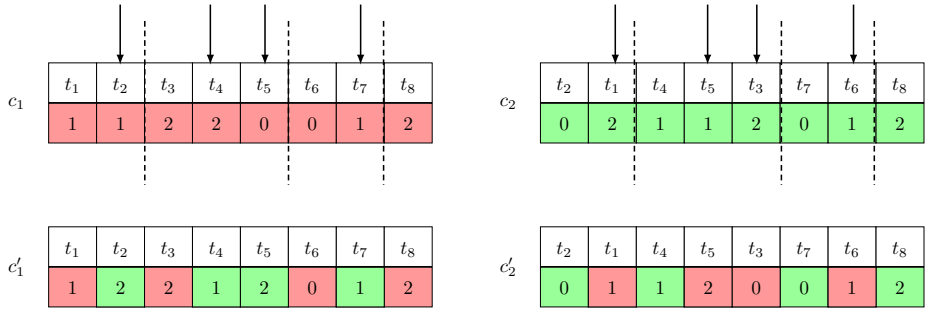
Figura 63 – Recombinação de Ponto Único com Partições na MSE.



Fonte: adaptado de Lee e Chen (2003).

complexo para o projeto de um GA (LEE; CHEN, 2003). Na recombinação de pontos Aleatórios (RPX[43]), um conjunto de tarefas é aleatoriamente selecionado. Cada tarefa deste conjunto define um ponto onde as alocações serão trocadas. A Figura 64 apresenta um exemplo do operador.

Figura 64 – Recombinação de Pontos Aleatórios.



Fonte: adaptado de Lee e Chen (2003).

Recombinação Uniforme (UX)

A UX também é um operador geralmente aplicado em diversos cenários envolvendo GAs. Neste caso, o trabalho de Yao, You e Li (2004) aplica o operador nos dois GAs propostos, sendo que ambos utilizam uma variação da MSE com TD e baseada em índice de prioridade para escalonamento. Em resumo, dado dois cromossomos selecionados, o operador constrói os cromossomos descendentes selecionando aleatoriamente cada gene dos pais. Em outras palavras, para cada gene de um cromossomo filho, lança-se uma moeda para definir de qual pai o gene será herdado.

[43] Do inglês, *Random Point Crossover*.

Recombinação de Ponto Único com Sequência Principal (MSSPX)

No segundo algoritmo apresentado, Yao, You e Li (2004) propõem operadores que utilizem os conceitos de MS e DAGMS (Subseção 2.1.1). Em resumo, $MS(t)$ refere-se, dentre todos os caminhos possíveis entre um nó de entrada à t, o caminho com a maior soma de custo computacional. Em sequência, dentre todas as possibilidades de MSs, DAGMS é a sequência com a mais alta soma de custo computacional. Utilizando esses conceitos, ambos os operadores, recombinação e mutação, são esforços para melhorar o desempenho do escalonamento utilizando duplicação de nós do DAGMS.

Como recombinação, dois operadores são designados. Na cadeia com os valores de prioridade, o operador UX é aplicado. Paralelamente, na cadeia de alocação, o SPX baseado em MS é executado (MSSPX[44]). O Algoritmo 26 enumera as instruções dessa recombinação, onde n é o conjunto de tarefas do grafo.

Algoritmo 26 Recombinação de Ponto Único baseado em MS – adaptado de Yao, You e Li (2004)

entrada: Cromossomos c_1 e c_2;
saída: Cromossomos c'_1 e c'_2;
1: Selecione aleatoriamente uma tarefa t;
{Gerando c'_1 e c'_2}
2: Troque todos os processadores alocados de todas as tarefas na MS de t entre os dois pais;
3: Troque os valores de prioridade de todas as tarefas entre os dois pais com uma certa prioridade, por exemplo, $1/n$;
4: **retorne:** c'_1 e c'_2

Recombinação Mapeada Ponderada (WMX)

A Recombinação Mapeada Ponderada (WMX[45]) é um dos operadores aplicados nas cadeias de escalonamento no trabalho de Bonyadi e Moghaddam (2009). Semelhante a recombinação de dois pontos, os pontos de corte, definidos aleatoriamente, delimitam uma subcadeia em cada cromossomo pai, onde cada subcadeia tem o seu conteúdo e ordem analisadas. A ordem da subcadeia adotada no primeiro pai é usada para reordenar a subcadeia correspondente ao segundo pai e vice-versa (BONYADI; MOGHADDAM, 2009).

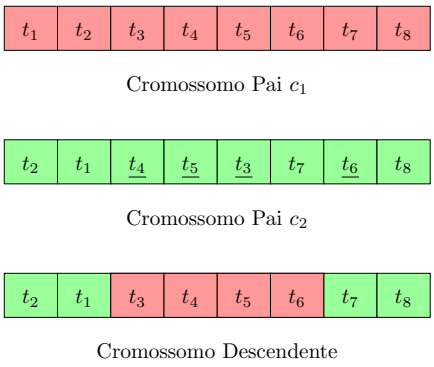
Recombinação de Ordem (OX)

O segundo operador aplicado nas cadeias de escalonamento no trabalho de Bonyadi e Moghaddam (2009) é uma Recombinação de Ordem (OX[46]). Esse operador realiza uma

[44] Do inglês, *Main Sequence Single Point Crossover*.
[45] Do inglês, *Weight Mapping Crossover*.
[46] Do inglês, *Order Crossover*.

permutação entre os genes do primeiro e segundo pai. Inicialmente, dois pontos de corte definem uma subcadeia no primeiro pai que é transmitida ao cromossomo descendente. Os genes dispostos na subcadeia são marcados no segundo pai. Em seguida, os genes não marcados no segundo pai são transmitidos ao cromossomo descendente seguindo a mesma ordenação. A Figura 65 apresenta um exemplo dessa recombinação aplicada em cadeias de escalonamento. Essa recombinação também está presente nos quatro operadores adotados por Dhingra, Gupta e Biswas (2014) e no trabalho de Omara e Arafa (2009).

Figura 65 – Recombinação de Ordem aplicada ao MSE.



Fonte: do autor.

Recombinação de Dois Pontos (TPX)

Ao adotar MSE, Gupta, Kumar e Agarwal (2010) implementam o operador TPX. Uma vez que as cadeias de um cromossomo possuem características distintas, a recombinação é aplicada de forma diferente em cada segmento, alocação e escalonamento. Inicialmente, o operador define o intervalo de corte entre x e y para um par de cromossomos selecionados. Em seguida, de forma aleatória, o operador define em qual cadeia a recombinação será aplicada. Se a cadeia de escalonamento for selecionada, cada subcadeia formada entre os pontos de corte é reordenada baseando-se no cromossomo oposto. Semelhantemente, se a cadeia de alocação for selecionada, as subcadeias são trocadas entre os pares selecionados na recombinação.

Chitra et al. (2010) é um exemplo de outro trabalho que utiliza a recombinação de dois pontos. No segundo GA do trabalho de Ahmad, Munir e Nisar (2012), a recombinação de dois pontos é antecedida por uma recombinação de um único ponto. Semelhante, Ahmad et al. (2016) emprega os operadores SPX e TPX, sendo que, ao final, efetua uma comparação para escolher os melhores cromossomos frutos das recombinações. Morady e Dal (2016) aplica o TPX exclusivamente na cadeia de alocação.

Recombinação de Ponto Único com Atividade (SPX)

O trabalho de Kang, Zhang e Chen (2011) implementa uma variação do operador SPX. Neste cenário, o ponto de corte é definido através da soma de uma probabilidade aleatória e do valor de atividade da tarefa (Expressão 28), de forma que tarefas são priorizadas baseando-se na possibilidade de melhora ou piora de desempenho. Conforme a definição de Kang, Zhang e Chen (2011), a Recombinação de Ponto Único com Atividade (ASPX[47]) possui três passos principais:

- ❏ Selecione uma subcadeia entre o primeiro gene g_i e o gene g_k, de forma que pb_i seja uma probabilidade randômica e $g_k = \max_{i=1...n}(ac_i + pb_i)$;
- ❏ Crie um cromossomo descendente ao copiar a subcadeia na mesma posição do cromossomo pai;
- ❏ As demais posições não preenchidas do cromossomo descendente devem ser substituídas observando todas as tarefas do segundo cromossomo, da esquerda à direita. Caso o gene já esteja disposto no descendente, o próximo gene será avaliado. Caso contrário, insira o gene do segundo cromossomo no cromossomo descendente.

Recombinação Parcialmente Combinada (PMX)

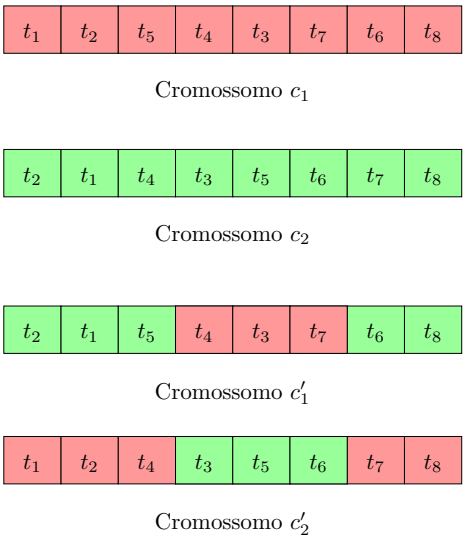
A recombinação Parcialmente Combinada (PMX) é um operador amplamente aplicado em codificações que utilizam permutações de valores. Empregado em um par de indivíduos, c_1 e c_2, o operador destaca um segmento delimitado por dois pontos de corte. Em seguida, utilizando o segmento de c_1, um cromossomo descendente é gerado. A próxima etapa é adaptação da permutação, onde uma comparação entre os segmentos de c_2 e de c_1 identifica os genes não copiados. Para cada gene não copiado presente em um segmento de c_2, é identificado o gene adjacente no segmento de c_1. Por fim, os genes não copiados são inseridos nas mesmas posições identificadas em c_2. Quando a posição já está preenchida, opta-se pela próxima posição do mapeamento. Após a identificação e correção de genes duplicados, o operador preenche o restante da cadeia do descendente com os genes de c_2. Para o segundo descendente, o processo é repetido ao inverter a ordem de aplicação nos cromossomos c_1 e c_2.

A Figura 66 apresenta um exemplo da aplicação do operador, onde dado dois cromossomos, c_1 e c_2, dois novos cromossomos descendentes são gerados, c'_1 e c'_2. A Figura 66(a) ilustra os cromossomos antes e após a aplicação do operador. Paralelamente, a Figura 66(b) relaciona os mapeamentos gerados nas etapas da recombinação.

Naturalmente aplicada na cadeia de escalonamento, Dhingra, Gupta e Biswas (2014) e Morady e Dal (2016) são alguns exemplos de trabalhos que empregaram o operador.

[47] Do inglês, *Activity Single Point Crossover*.

Figura 66 – Recombinação Parcial Mapeada na MSE.



(a) Cromossomos recombinados.

Descendente c'_1:
$t_5 \rightarrow t_3 \rightarrow t_4$
$t_6 \rightarrow t_7$

Descendente c'_2:
$t_4 \rightarrow t_3 \rightarrow t_5$
$t_7 \rightarrow t_6$

(b) Relação de Mapeamento.

Fonte: do autor.

Entretanto, o operador pode propagar mudanças que não atendem as restrições de precedência. Portanto, para populações com cromossomos factíveis, a aplicação do operador deve ser amparado por algum mecanismo reparador ou eliminador de soluções inválidas (BONYADI; MOGHADDAM, 2009).

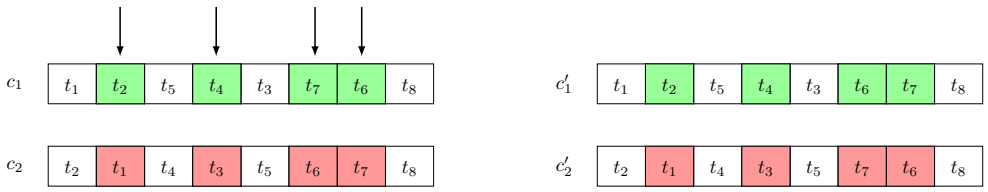
Recombinação de Posição (PX)

Outra recombinação aplicada em codificações estruturadas em permutação é a Recombinação de Posição (PX[48]), sendo um operador aplicado em Dhingra, Gupta e Biswas (2014). Neste método, dado um par de cromossomos, uma quantidade k de genes é definida aleatoriamente para troca. Em seguida, para formar o primeiro filho, os k genes selecionados do primeiro pai são reordenados baseando-se na ordem em que esses genes são encontrados no segundo pai. Os demais genes são transmitidos diretamente ao descendente. O segundo filho é gerado semelhantemente, porém com uma ordem inversa de aplicação nos cromossomos pais.

A Figura 67 ilustra um exemplo da aplicação do operador. Neste cenário, para o primeiro filho, c'_1, a sequência de k tarefas recebida pelo cromossomo pai, c_1, é t_2, t_4, t_7 e t_6. Posteriormente, esta sequência é reordenada baseando-se na ordem do segundo pai, c'_2, isto é, t_2, t_4, t_6 e t_7. O segundo filho, c'_2, também reordena seus k elementos, todavia, sendo tarefas diferentes.

[48] Do inglês, *Position Crossover*.

Figura 67 – Recombinação de Posição na MSE.



Fonte: do autor.

Da mesma forma que o operador PMX, a recombinação baseada em posição também pode propagar cromossomos inválidos. Sendo assim, neste cenário, conforme supracitado, é importante a aplicação de algum mecanismo eliminador ou reparador de soluções inválidas.

4.3.3.2 Mutações

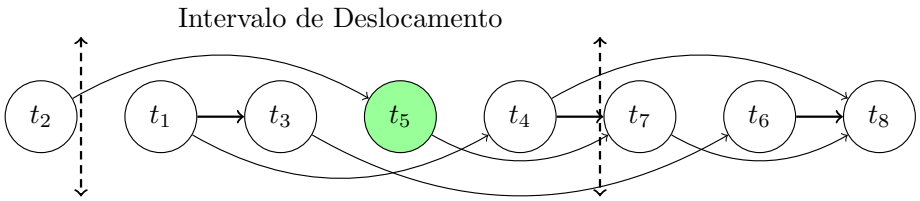
Com a utilização das cadeias de escalonamento e alocação, a MSE possui diversas opções de mutação. A primeira subseção contextualiza uma mutação que desloca tarefas. As seguintes subseções apresentam mutações baseadas em definição de novos valores de alocação, alteração dos índices de prioridade, utilização de sequência principal, troca, embaralhamento, inversão e emprego do conceito de atividade do gene.

Mutação de Deslocamento de Tarefas (STM)

Um dos operadores de mutação propostos nos trabalhos de Wang, Siegel e Roychowdhury (1996), Wang et al. (1997) e Gupta, Kumar e Agarwal (2010) é a Mutação de Deslocamento de Tarefas (STM[49]). Sendo aplicado na cadeia de escalonamento, STM seleciona uma tarefa aleatoriamente e, em seguida, define uma nova posição para a mesma na codificação. A nova posição, também definida aleatoriamente, deve ser limitada às restrições de precedência. Wang, Siegel e Roychowdhury (1996) definem o intervalo para o deslocamento ao observar as arestas de precedência dispostas no seu modelo de cadeia de escalonamento: depois da mais próxima aresta que envia dados a tarefa selecionada e antes da mais próxima aresta que envia os dados desta. Gupta, Kumar e Agarwal (2010), Chitra, Venkatesh e Rajaram (2011) e Sathappan et al. (2011) definem o intervalo de troca da tarefas selecionada entre as suas tarefas predecessoras e sucessoras imediatas. A Figura 68 traz um exemplo com o intervalo de deslocamento para a tarefa t_5.

[49] Do inglês, *Shift Task Mutation*.

Figura 68 – Mutação de Deslocamento de Tarefas na MSE.



Fonte: adaptado de Wang, Siegel e Roychowdhury (1996).

Mutação de Processador (PM)

Outro operador aplicado em Wang, Siegel e Roychowdhury (1996) é o PM. Naturalmente aplicado na cadeia de alocação, PM define aleatoriamente um novo processador para uma tarefa da lista. Esse operador também é uma opção aplicada nos trabalhos de Lee e Chen (2003), Yao, You e Li (2004), Omara e Arafa (2009), Chitra et al. (2010) Gupta, Kumar e Agarwal (2010), Sathappan et al. (2011), Kaur e Singh (2012) e Morady e Dal (2016). Nos trabalho de Bonyadi e Moghaddam (2009) e Dhingra, Gupta e Biswas (2014), na cadeia de alocação, esse mesmo operador também é aplicado, porém denominando-se mutação uniforme. O trabalho de Chitra, Venkatesh e Rajaram (2011) apresenta uma variação do operador PM ao destinar a alocação ao processador com o maior nível de confiabilidade. A cadeia de alocação também é modificada por Ahmad, Munir e Nisar (2012) ao aplicar mutações em um ou dois pontos de alteração. De forma análoga, a alteração de um ou dois pontos da cadeia de alocação é empregada por Ahmad et al. (2016) com um mecanismo de comparação para escolha dos melhores indivíduos originados na mutação.

Mutação de Probabilidade Aleatória (RPM)

Utilizando uma versão adaptada da MSE, que codifica a prioridade de tarefas, Yao, You e Li (2004) propõe um operador de mutação semelhante ao MP, contudo, a modificação é executada nos genes que identificam a prioridade. Desta forma, em resumo, genes selecionados aleatoriamente pelo operador têm um novo valor de prioridade definido entre 1 e MAX_NUM. Portanto, a Mutação de Probabilidade Aleatória (RPM[50]) é exclusivamente aplicada na cadeia de escalonamento.

[50] Do inglês, *Random Probability Mutation*.

Mutação de Troca com Sequência Principal (MSSM)

Conforme apresentado na Subseção 4.3.3.1, Yao, You e Li (2004) propõem operadores baseados em MS e DAGMS. Como funcionamento da Mutação de Troca com Sequência Principal (MSSM[51]), caso a cadeia de alocação seja selecionada, o Algoritmo 27 é executado, caso contrário, a RPM é executada (Subseção 4.3.3.2). Segundo a definição de Yao, You e Li (2004), para trocar o processador alocado, o algoritmo deve verificar se a tarefa selecionada está presente no DAGMS ou não. Dessa forma, se a tarefa estiver presente no DAGMS, um processador é aleatoriamente selecionado. Este novo processador é adicionado caso a tarefa não esteja alocada nele. Em contrapartida, o algoritmo elimina a alocação do processador para a tarefa. Por fim, caso a tarefa esteja com zero processadores, um novo processador é aleatoriamente alocado.

Algoritmo 27 Troca de processador baseado em MS – adaptado de Yao, You e Li (2004)

entrada: Cromossomo c;
saída: Cromossomo c';
1: Selecione uma tarefa t do cromossomo c;
{Altere a prioridade (linha 3) ou o processador alocado (linha 5)};
2: Gere um novo valor de prioridade para a tarefa t;
3: **se** tarefa t está em DAGMS então **então**
4: Troque seu valor de processador;
5: **senão**
6: Considere t como um tarefa em DAGMS com uma pequena possibilidade ou aleatoriamente aloca um processado para t.
7: **fim se**
8: **retorne:** c'

Mutação de Troca (SM), Embaralhamento (SFTM) e Inversão (IM)

Sobre a cadeia de escalonamento, Bonyadi e Moghaddam (2009) e Dhingra, Gupta e Biswas (2014) utilizam três operadores de mutação distintos: Mutação de Troca (SM), Mutação de Embaralhamento de Tarefas (SFTM[52]) e Mutação de Inversão (IM[53]) :

- **Troca:** dois genes são selecionados e, em seguida, seus conteúdos são trocados;
- **Embaralhamento:** uma subcadeia formada com dois pontos selecionados aleatoriamente tem seu conteúdo embaralhado;
- **Inversão:** o conteúdo de uma subcadeia formado com dois pontos selecionados aleatoriamente é invertido.

[51] Do inglês, *Main Sequence Swap Mutation.*
[52] Do inglês, *Shuffle Task Mutation.*
[53] Do inglês, *Invert Mutation.*

Chitra et al. (2010), Mohamed e Awadalla (2011), Panwar, Lal e Singh (2012), Awadall, Ahmad e Al-Busaidi (2013) e Morady e Dal (2016) são outros exemplos de trabalhos que utilizas a SM.

Mutação de Troca com Processadores (PSM)

Aplicado na cadeia de alocação, a Mutação de Troca de Processadores (PSM) efetua a troca de alocações entre um par de genes selecionados aleatoriamente. Uma vez que a cadeia de alocação não apresenta maiores restrições de ordenação, a PSM pode ser aplicada sem o uso de mecanismos reparadores. Alguns exemplos de trabalhos que utilizam a PSM são Lee e Chen (2003), Mohamed e Awadalla (2011), Panwar, Lal e Singh (2012) e Awadall, Ahmad e Al-Busaidi (2013)

Mutação de Processador com Atividade (APM)

De forma semelhante a Recombinação Baseada em Atividade (Subseção 4.3.3.1), Kang, Zhang e Chen (2011) também definem um operador de mutação que utiliza sua medida de atividade (Expressão 28): Mutação de Processador com Atividade (APM[54]). Aplicado à cadeia de alocação, o operador analisa se uma tarefa deve ter seu processador modificado. A realocação é aplicada caso a soma de uma probabilidade aleatória e o valor de atividade de uma tarefa forem maiores que um determinado limiar. De acordo com Kang, Zhang e Chen (2011), tarefas que possam ter o desempenho melhorado recebem uma probabilidade maior.

4.3.4 Codificação por Tuplas (TE)

A Codificação por Lista de Tuplas (TE[55]) reúne em cada gene uma combinação entre a identificação da tarefa juntamente ao processador ao qual ela será alocada. Desta forma, conforme a descrição de Topcuoglu e Sevilmis (2002), cada gene de um cromossomo c é uma tupla, sendo que $c(i) = (t_j, p_k)$ identifica no i-ésimo gene, a tarefa t_j e seu respectivo processador p_k. O comprimento do cromossomo é denotado por n, onde n é a quantidade de tarefas no grafo. A codificação ainda aborda as restrições de precedência, configuradas aqui como a ordem em que as tarefas estão dispostas no cromossomo. Portanto, se o grafo de tarefas possui uma aresta (t_i, t_j), então t_i deve vir antes de t_j no cromossomo (TOPCUOGLU; SEVILMIS, 2002). A Figura 69 apresenta um cromossomo em TE baseado no grafo de tarefas da Figura 22(a) e em um cenário com três processadores.

Com uma codificação similar à TE, onde os genes identificam tuplas (t_n, p_m) combinam identificação tarefa e processador alocado, Wu et al. (2004) implementam uma

[54] Do inglês, *Activity Processor Mutation*.
[55] Do inglês, Tuple Encoding.

Figura 69 – Codificação de Tuplas.

Índices	1	2	3	4	5	6	7	8
Tarefas	t_2	t_1	t_3	t_5	t_4	t_7	t_6	t_8
Processadores	1	2	1	2	3	1	1	2

Fonte: adaptado de Topcuoglu e Sevilmis (2002).

variação com diminuição das restrições da representação. A primeira diferença está no comprimento do indivíduo, sendo variável nesta versão. Isso ocorre pois um cromossomo pode não representar todas as tarefas do grafo. Além disso, alguns genes podem identificar tarefas alocadas para mais de um processador, configurando, assim, um cenário com TD. Se mais de um gene estiver representando a mesma tarefa e alocação, será considerado somente o primeiro gene disposto no cromossomo, considerando uma leitura da esquerda para a direita. Apesar de, na população inicial, todos os cromossomos apresentarem o mesmo comprimento e representarem todas as tarefas, após a execução de recombinações e mutações, os cromossomos podem sofrer grandes modificações. A terceira codificação adotada no trabalho de Omara e Arafa (2009) apresenta as mesmas características da codificação TE com redução de restrições e duplicação de tarefas. Quanto às restrições de precedência, Wu et al. (2004) alertam que cromossomos, que apresentam ordenação inválida de tarefas, têm seu valor de aptidão penalizado na função objetivo.

A Figura 70 ilustra dois exemplos da codificação baseada em TE com diminuição de restrições. O primeiro cromossomo, disposto na Figura 70, apresenta duplicação de tarefas, onde a tarefa t_2 está alocada aos processadores p_4 e p_3. Além disso, a tarefa t_4 está duplicada no mesmo processador, o que indica que somente a alocação mais à esquerda do cromossomo será codificada à solução. Em contrapartida, o segundo cromossomo, na Figura 70, apresenta um comprimento reduzido. Nessa codificação, um cromossomo pode ter comprimento variado após a realização das recombinações e mutações. Desta forma, nem todas as tarefas do grafo são codificadas, portanto, uma codificação que lida com cromossomos infactíveis.

Figura 70 – Codificação baseada em TE com diminuição de restrição.

Índices	1	2	3	4	5	6	7	8	9
Tarefas	t_4	t_2	t_7	t_2	t_4	t_5	t_6	t_1	t_3
Processadores	1	4	3	3	1	4	3	1	2

1	2	3	4
t_4	t_2	t_2	t_3
1	4	4	3

Fonte: adaptado de Wu et al. (2004).

A codificação adotada no trabalho de Amalarethinam e Selvi (2012) também é uma variante da TE, em que duplicação de tarefas e cromossomos com tamanho variável são permitidos. Contudo, diferente da codificação do trabalho de Wu et al. (2004), um cromossomo deve conter todas as tarefas do grafo, de forma que soluções inválidas são descartadas das futuras gerações. Além disso, a ordem em que as tarefas estão dispostas no cromossomo reflete a sequência de execução. Sendo assim, esse formato adotado por Amalarethinam e Selvi (2012) emprega variações de recombinações e mutações voltadas a reparar soluções inválidas.

De forma análoga, a codificação adotada por Amirjanov e Sobolev (2017) também considera uma codificação baseada em TE, todavia com cromossomos válidos e inválidos de tamanho fixo, que também apresentam a ordenação das tarefas refletida na ordenação dos genes. A diferença, quanto ao trabalho de Amirjanov e Sobolev (2017), é a implementação de um operador local exclusivo de reparação com o objetivo de recuperar possíveis soluções válidas.

Por fim, os trabalhos de Akbari, Rashidi e Alizadeh (2017) e Akbari (2018) implementam a codificação baseada em TE dividindo um escalonamento em diversos segmentos. Ao controlar essas divisões, a codificação identifica as possíveis posições, dentro de um cromossomo, em que as tarefas podem ser deslocadas sem violar as restrições de precedência. Dessa forma, dentro de cada segmento, haverá um conjunto de tarefas independentes, isto é, tarefas propícias a serem executadas paralelamente.

4.3.4.1 Recombinações

Com diversas adaptações da TE, variadas recombinações podem ser aplicadas. A primeira subseção apresenta uma combinação de ordenação e Recombinação de Ponto Único. Uma recombinação com menor restrição e baseada em vários pontos de corte é introduzida na subseção seguinte. De forma análoga, uma recombinação com um mecanismo verificador de soluções também é apresentada. Por fim, as demais subseções contextualizam versões da recombinação com dois pontos de corte.

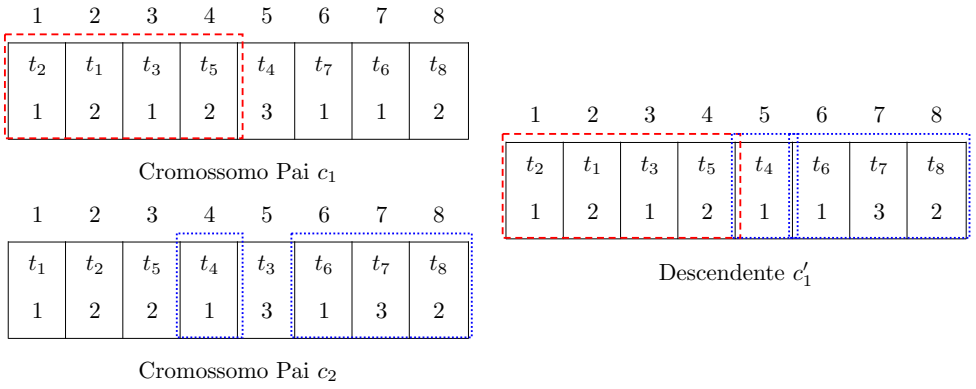
Recombinação de Ponto Único com Ordenação (SPX)

A implementação de Topcuoglu e Sevilmis (2002) descreve uma recombinação que divide pares de cromossomos utilizando um único ponto de corte (SPX). Em seguida, a transmissão dos genes de cada cromossomo é realizada considerando a ordem dos genes em cada cadeia. Além de implementarem o operador, os trabalhos de Akbari, Rashidi e Alizadeh (2017) e Akbari (2018) demonstram a exatidão, completude e singularidade dessa recombinação. Para a geração de um novo cromossomo descendente, dividi-se o par de cromossomos com um ponto de corte definido aleatoriamente. Em seguida, a metade à esquerda do primeiro pai é inteiramente copiada ao cromossomo filho. Os genes restantes

do cromossomo filho são coletados e anexados na mesma ordem em que são encontrados no segund pai. Para a geração de um segundo filho, a estratégia é realizada da mesma forma, porém invertendo a ordem dos pais. Segundo Topcuoglu e Sevilmis (2002), essa estratégia preserva as restrições de precedência ao não gerar escalonamentos inválidos.

A Figura 71 apresenta um exemplo de aplicação do SPX com Ordenação em dois cromossomo válidos. A metade à esquerda do cromossomo c_1, $\{t_2, t_1, t_3, t_5\}$, é transmitida diretamente ao cromossomo descendente. As demais tarefas que restam, t_4, t_6, t_7 e t_8, são copiadas, ao cromossomo filho, na mesma ordem em que aparecem no cromossomo c_2.

Figura 71 – Recombinação de Ponto Único com Ordenação na TE.



Fonte: adaptado de Topcuoglu e Sevilmis (2002).

Demiroz e Topcuoglu (2006) é um outro exemplo de trabalho que utiliza o operador SPX com ordenação.

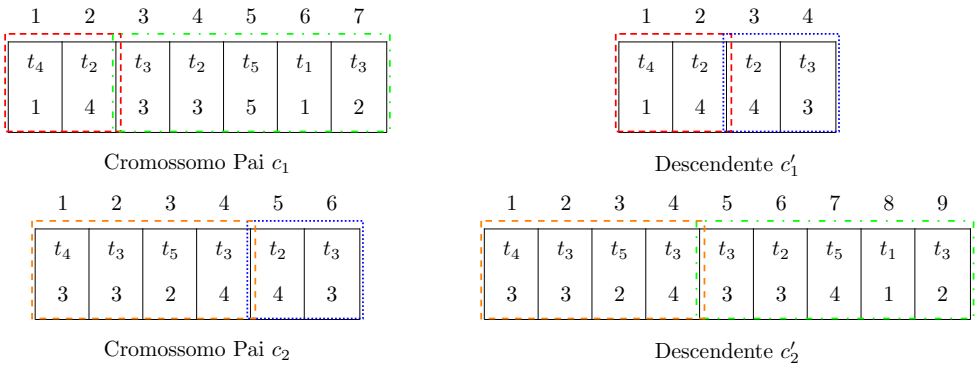
Recombinação Multiponto (MPX)

O trabalho de Wu et al. (2004) apresenta uma variante do SPX. Ao empregar a codificação baseada na TE com redução de limitações, o MPX não é limitado às restrições de ordem. Desta forma, o operador é simplificado ao conduzir somente a troca do material. Outra diferença empregada é representada pelo sorteio do ponto de corte, onde, nesta versão, os pontos de corte são sorteados para cada indivíduo. As demais instruções seguem o formato da recombinação de um ponto: a metade à esquerda do ponto de corte do primeiro pai é copiada ao filho; o restante do primeiro filho é formado pela metade à direita do ponto de corte do segundo pai; o processo invertido é repetido para a geração do segundo filho.

Um exemplo do operador MPX é ilustrado na Figura 72. O ponto de corte do cromossomo c_1 está definido entre os índices 1 e 2, enquanto que o ponto de corte do cromossomo c_2, entre 4 e 5. Com os pontos de corte definidos, o material é separado e trocado. Os

filhos gerados, descendente c'_1 e c'_2, possuem comprimento variado e obedecem as divisões de cada ponto de corte dos pais.

Figura 72 – Recombinação Multiponto.



Fonte: adaptado de Wu et al. (2004).

Recombinação de Ponto Único com Verificação (SPX)

A variação na codificação empregada por Amalarethinam e Selvi (2012) exige que todas as tarefas do grafo sejam codificadas. Dessa forma, a recombinação adotada efetua uma verificação nos cromossomos descendentes para identificar soluções inválidas. Já o funcionamento do operador adota um SPX básico, onde um único ponto de corte é utilizado para dividir os cromossomos pais e permitir a separação e troca do material genético. Indivíduos com soluções inválidas são encaminhados para a etapa de mutação.

Recombinação de Dois Pontos (TPX)

Em um cenário com redução de restrições e TD, a terceira codificação empregada por Omara e Arafa (2009) implementa um operador de recombinação simples baseado em dois pontos de corte (TPX). Após a aplicação da recombinação, e conforme as caraterísticas desta versão da codificação, cromossomos recombinados podem apresentar comprimento variado, tarefas não codificadas e duplicação de alocações. No operador, dois pontos são aleatoriamente escolhidos e as partições entre esses pontos são trocadas entre dois cromossomos pais para formar dois novos cromossomos (OMARA; ARAFA, 2009).

Recombinação de Dois Pontos com Ordenação (TPX)

Uma variante do SBX com ordenação é empregado no trabalho de Amirjanov e Sobolev (2017). Nesta versão, dentre um par de cromossomos, dois pontos de corte aleatórios são utilizados (TPX), de modo que o material genético disposto entre esses dois pontos é

transmitido, via o primeiro pai, para o cromossomo descendente. Os demais genes, além dos limites dos pontos de corte, são copiados obedecendo a ordem de disposição na cadeia do cromossomo do segundo pai. Para a geração de um segundo cromossomo descendente, o operador é reaplicado com a ordem inversa dos pais.

4.3.4.2 Mutações

Variando a aplicação em cenários com maior ou menor restrição, a TE possui diversas opções de mutação. A primeira subseção apresenta uma mutação baseada no deslocamento de tarefas. Uma mutação de substituição dos valores de alocação é contextualizada na subseção seguinte. Em um cenário com menores restrições, uma mutação de redefinição de tarefas e processadores também é introduzida. Por fim, ao final da seção, é apresentada uma mutação orientada à recuperação de soluções infactíveis.

Mutação de Deslocamento de Tarefas (STM)

Os experimentos no trabalho de Topcuoglu e Sevilmis (2002) adotam três variações de mutação, tipo 1, tipo 2 e tipo 3, sendo respectivamente, Mutação de Deslocamento de Tarefas (STM), Mutação de Processador (PM) e uma combinação de ambas.

O operador STM define uma nova posição para a tarefa selecionada aleatoriamente. A nova posição não pode invalidar a precedência entre as tarefas, de forma que um intervalo de transição é definido para que se possa atender essa restrição. Dada uma tarefa t_i, onde $t_j \succ t_i \succ t_k$, isto é, t_j identifica as tarefas precedentes à t_i, enquanto que t_k identifica as tarefas descendentes de t_i; o início do espaço do intervalo de transição é definido como a primeira tarefa precedente, $t_j + 1$; e o segundo espaço do intervalo é atribuído à primeira tarefa descendente, $t_k - 1$. Portanto, conforme o exemplo da Figura 73 (a) e 73 (b), a tarefa t_4 pode ser deslocada entre os índices 2 e 6. Ao identificar os segmentos onde uma tarefa pode ser movida como áreas de mutação, os trabalho de Akbari, Rashidi e Alizadeh (2017) e Akbari (2018) destacam a exatidão da aplicação do operador STM. Esse operador também é aplicado no trabalho de Demiroz e Topcuoglu (2006).

Mutação de Processador (PM)

A Mutação de Processador (PM) altera, aleatoriamente, a unidade a qual uma tarefa estava alocada. A Figura 73 (c) ilustra um exemplo do operador por sorteio de processador. O trabalho de Omara e Arafa (2009) também aplica uma mutação com sorteio de processadores, todavia em uma codificação com menos restrições, onde alocações e tarefas podem ser duplicadas.

O trabalho de Demiroz e Topcuoglu (2006) utiliza o operador PM em um cenário sem TD, de forma só exista uma unidade de alocação para cada tarefa.

Figura 73 – Combinação das Mutações de Processador e Deslocamento de Tarefas.

1	2	3	4	5	6	7	8
t_2	t_1	t_3	t_5	t_4	t_7	t_6	t_8
1	2	1	2	3	1	1	2

(a) Intervalo válido para deslocamento da tarefa t_4.

1	2	3	4	5	6	7	8
t_2	t_1	t_3	t_4	t_5	t_7	t_6	t_8
1	2	1	3	2	1	1	2

(b) Aplicação do operador de deslocamento.

1	2	3	4	5	6	7	8
t_2	t_1	t_3	t_4	t_5	t_7	t_6	t_8
1	2	1	1	2	1	1	2

(c) Aplicação do operador de sorteio de processador.

Fonte: do autor.

Mutação de Tarefa (TM) ou Processador (PM)

Considerando a variação da codificação baseada em TE, onde o cromossomo pode ter tamanho variado, tarefas podem ser duplicadas e nem todas as tarefas podem estar codificadas, o trabalho de Wu et al. (2004) implementa um operador de mutação que explora essas expansões. Quando um gene é selecionado, a alocação do processador ou a identificação da tarefa podem ser alteradas aleatoriamente. Portanto, o mesmo operador assume o modelo de Mutação de Tarefa (TM[56]) ou Processador (PM). Amirjanov e Sobolev (2017) é um outro exemplo de aplicação da PM.

Mutação de Recuperação (RCM)

O operador de Mutação de Recuperação (RCM[57]) adotado por Amalarethinam e Selvi (2012) é utilizado na tentativa de recuperar algum cromossomo que codifica uma solução inválida. Sendo assim, uma troca entre as tarefas pode ser efetuada na tentativa de atender alguma restrição de precedência. O operador também pode anexar tarefas que estejam faltando no cromossomo. Ao final da mutação, o cromossomo é reavaliado e caso seja factível, é enviado à próxima geração.

[56] Do inglês, *Task Mutation*.
[57] Do inglês, *Recovery Mutation*.

4.3.4.3 Operador de Inversão

Conforme citado anteriormente, os trabalhos de Akbari, Rashidi e Alizadeh (2017) e Akbari (2018) implementam uma codificação baseada em TE que mantém um controle de segmentos, onde as tarefas podem ser deslocadas sem problemas quanto à precedência. Esses segmentos também identificam tarefas independentes que possam ser processadas paralelamente. Dessa forma, utilizando os espaços dos segmentos, um operador adicional foi empregado. O operador de inversão altera a ordem de todas as tarefas de um segmento, de modo que as tarefas dispostas ao final do segmento são realocadas ao início do espaço e que a ordem de alocação de processadores não seja alterada. O Algoritmo 28 sumariza a aplicação desse operador. Em adição, a Figura 74 apresenta uma aplicação prática em um cromossomo segmentado.

Algoritmo 28 Inversão da Segmentação – adaptado de Akbari, Rashidi e Alizadeh (2017)

entrada: População atual;
saída: Nova população invertida;
1: **para todo** cromossomo na população **faça**
2: **para todo** nível dentro do cromossomo atual **faça**
3: Reverta a ordem de todas as tarefas;
4: **fim para**
5: **fim para**
6: **retorne:** Nova população invertida;

Figura 74 – Operador de inversão baseado em segmentos.

Índices	1	2	3	4	5	6	7	8
Tarefas	t_2	t_1	t_3	t_4	t_5	t_7	t_6	t_8
Processadores	1	2	1	2	3	1	1	2

(a) Segmentos baseados no DAG.

Índices	1	2	3	4	5	6	7	8
Tarefas	t_1	t_2	t_5	t_4	t_3	t_6	t_7	t_8
Processadores	1	2	1	2	3	1	1	2

(b) Aplicação do operador.

Fonte: adaptado de Akbari, Rashidi e Alizadeh (2017).

4.4 Outras Representações

Esta seção apresenta a quarta categoria de representações. Nesta divisão, são agrupadas as codificações com aspectos muito específicos de seus respectivos cenários. Por exemplo, representações que codificam a capacidade de permutação de frequência e desempenho dos processadores. Outras representações divergentes também são listadas nesta seção. Desse modo, primeiramente, a Codificação por Lista de Mapeamentos é descrita na Subseção 4.4.1. Em seguida, na Subseção 4.4.2, a Codificação de Duplo nível é descrita. Logo após, a Codificação de Processadores com recursos de DVFS é apresentada na Subseção 4.4.3. Por fim, a última representação apresentada é a Codificação de Q-bits, sendo essa contextualizada na Subseção 4.4.4.

4.4.1 Codificação por Lista de Mapeamentos (MLE)

A Codificação por Lista de Mapeamentos (MLE[58]) trabalha diretamente com alocações de máquinas em ambientes heterogêneos de computação. Proposta por Singh e Youssef (1996), a MLE relaciona possíveis modos de execução para cada máquina multiprocessadora designada. A ordenação topológica entre tarefas é reproduzida na função objetivo, de forma que a codificação contemple exclusivamente aspectos da alocação.

O cromossomo é um vetor de tamanho n, onde n é a quantidade de tarefas no grafo. Cada gene desse vetor é preenchido com algum número inteiro presente na base $r \times \phi$. Dessa forma, em um ambiente com r tipos de máquinas interconectadas por λ conexões, existem ϕ possíveis mapeamentos para cada tarefa. Assume-se o valor 1 para ϕ em ambientes de máquinas com processamento serial, isto é, máquinas diferentes de SIMD[59], MIMD[60], dentre outras (SINGH; YOUSSEF, 1996). Portanto, cada gene representa o mapeamento adotado e a máquina alocada à tarefa. Para a decodificação de cada gene presente no cromossomo, a Expressão 30 é aplicada, onde g_i identifica o gene de um cromossomo c, e π indica decodificação de cada gene:

$$\begin{aligned} \pi(i) = (([g_i/\phi] + 1), (g_i \mod \phi + 1)) \\ \forall i \in T.g_i \text{ é o i-ésimo gene de } c \end{aligned} \tag{30}$$

A Figura 75 contempla um exemplo da lista de mapeamentos. Construído sobre o grafo da Figura 22(a), o ambiente de processamento adotado assume $t = 2$ tipos de máquinas, onde cada estação apresenta $\phi = 3$ mapeamentos possíveis. Consequentemente, o cromossomo pode ser modelado como um vetor com genes de valores entre 0 e 5, dado

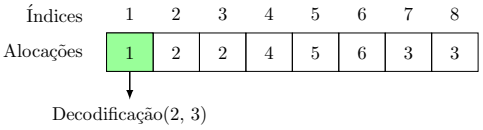
[58] Do inglês, *Mapping List Encoding.*

[59] *Single instruction, multiple data:* em uma ambiente de computação paralela, a mesma instrução é executada simultaneamente em múltiplos dados, de forma a explorar o paralelismo e não a concorrência.

[60] *Multiple instruction, multiple data:* um conjunto de processadores organizados de maneira a executar variadas instruções em diferentes segmentos de dados.

que cada gene é uma valor definido por $r \times \phi$. Conforme demonstrado nas recombinações, os genes não compõe uma permutação, ou seja, genes podem conter valores de alocação iguais, de tal forma que demonstrem a alocação de mais de uma tarefa para um mesmo conjunto máquina/mapeamento.

Figura 75 – Codificação construída por lista de mapeamentos.



Fonte: do autor.

4.4.1.1 Recombinações

A Figura 76 apresenta exemplos das recombinações empregadas na MLE utilizando de dois cromossomos (c_1 e c_2). Os operadores utilizados são as Recombinações de Ponto Único (SPX, Figura 76(a)), Dois Pontos (TPX), Figura 76(b)) e Uniforme (UX[61], Figura 76(c)). Nas recombinações de um e dois pontos, os elementos são trocados de acordo com o cortes sorteados. Em contrapartida, a recombinação uniforme define as trocas de genes baseando-se em uma máscara binária. As permutações geradas pelos operadores não formam indivíduos não-válidos, uma vez que a ordenação (principal restrição) é empregada na função objetivo.

4.4.1.2 Mutação de Mapeamento (MM)

De forma análoga à PM, a Mutação de Mapeamento (MM[62]) é caracterizada pelo sorteio de um gene que sofrerá uma reatribuição aleatória, o que, consequentemente, altera a alocação e o mapeamento definidos.

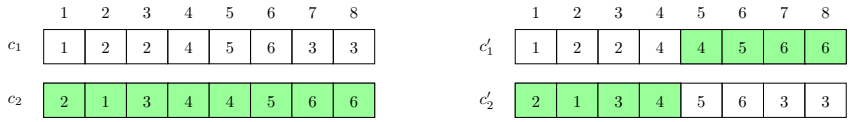
4.4.2 Codificação de Duplo Nível (DLE)

A Codificação de Duplo Nível (DLE[63]) é encontrada no trabalho de Liu, Li e Yu (2002). Representando um cenário muito específico do problema, a codificação contém um único modelo de recombinação e de mutação. A avaliação da aptidão é semelhante a outros modelos, onde o makespan é a métrica explorada.

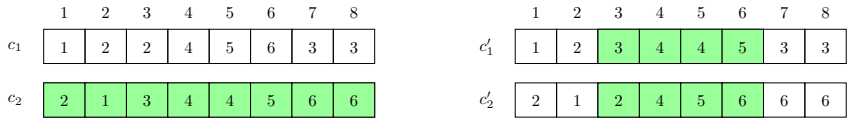
A codificação empregada no trabalho de Liu, Li e Yu (2002) trabalha sob o cenário onde um grafo de tarefas pode ser segmentado em pequenas novas subtarefas também

[61] Do inglês, *Uniform Crossover*.
[62] Do inglês, *Mapping Mutation*.
[63] Do inglês, *Double Level Encoding*.

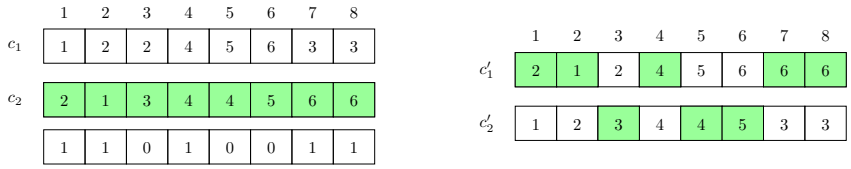
Figura 76 – Recombinações empregadas à MLE.



(a) Recombinação de um Único Ponto.



(b) Recombinação de Dois Pontos.



(c) Recombinação Uniforme.

Fonte: do autor.

relacionadas por precedência. Dessa forma, é proposta uma codificação sobre os seguintes critérios:

- ❏ As restrições de precedência entre as tarefas do grafo são obedecidas;
- ❏ Cada subtarefa aparece somente uma única vez no escalonamento;
- ❏ Os processadores são suficientes para a execução das subtarefas.

Em resumo, DLE reúne o mapeamento de tarefas para cada subrede e para cada nó de processamento. Conforme a definição de Liu, Li e Yu (2002), o primeiro nível da DLE contém LN listas que mapeiam as tarefas que executarão nas correspondentes subredes de processamento, enquanto que o segundo nível contém ln listas que indicam subtarefas alocadas aos nós de processamento. A disposição dos itens na lista corresponde a ordem de execução das tarefas e subtarefas. Desta forma, um escalonamento $S = (L_1, L_2, ..., L_{LN})(l_1, l_2, ..., l_{ln})$ pode ser representado em DLE, conforme o exemplo ilustrado na Figura 77.

4.4.2.1 Recombinação Interna (IX)

A recombinação interna (IX[64]), proposta por Liu, Li e Yu (2002) à DLE, efetua a troca de tarefas entre os nós de processamento. O Algoritmo 29 sumariza as instruções do IX:

[64] Do inglês, *Inner Crossover*.

Figura 77 – Codificação de Duplo Nível.

Primeiro nível da codificação:
Subrede → Tarefas
Segundo nível da codificação:
Processador → Subtarefas

(a) Modelo da Codificação.

Primeiro nível da codificação:
$N_1 \rightarrow T_1, T_3$
$N_2 \rightarrow T_2$
Segundo nível da codificação:
$C_{11} \rightarrow T_{11}, T_{12}, T_{32}, T_{35}$
$C_{12} \rightarrow T_{13}, T_{31}, T_{33}, T_{34}$
$C_{21} \rightarrow T_{21}, T_{23}$
$C_{22} \rightarrow T_{22}, T_{24}, T_{25}$

(b) Exemplo da Codificação.

Fonte: adaptado de Liu, Li e Yu (2002).

Algoritmo 29 Recombinação Interna – adaptado de Liu, Li e Yu (2002)

entrada: Cromossomo c;
saída: Cromossomo s';
1: Gere um número aleatório x;
2: Selecione aleatoriamente duas listas de um escalonamento;
3: Determine o ponto de corte de acordo com x;
4: Divida as duas listas em dois segmentos;
5: **se** as tarefas dos segmentos inferiores de uma lista podem ser executas em ordem correspondente ao nó de processamento **então**
6: Troque os segmentos inferiores das duas listas e gere um novo cromossomo c;
7: **fim se**
8: **retorne:** s';

Novos cromossomos gerados pelo IICX são válidos desde que corroborem com as seguintes condições:

- ❏ A altura de uma tarefa t_i próxima ao ponto de corte $altura(t_i) \neq h$;
- ❏ A altura de uma tarefa t_i anterior ao ponto de corte $altura(t_i) \leq h$.

4.4.2.2 Mutação de Listas (LM)

Liu, Li e Yu (2002) propõe o operador de Mutação de Listas (LM[65]) para efetuar trocas aleatórias entre tarefas com um mesmo valor de altura. O Algoritmo 30 apresenta as instruções da LM:

De forma análoga ao operador IX, LM produz cromossomos válidos se os seguintes critérios forem atendidos:

- ❏ A altura de uma tarefa t_i próxima ao ponto de mutação $altura(t_i) \geq h$;
- ❏ A altura de uma tarefa t_i anterior ao ponto de mutação $altura(t_i) \leq h$;

[65] Do inglês, *List Mutation*.

Algoritmo 30 Mutação de Listas – adaptado de Liu, Li e Yu (2002)

entrada: Escalonamento c;
saída: Escalonamento c';
1: Gere um número aleatório x;
2: Selecione aleatoriamente duas listas l_i e l_j de um escalonamento c;
3: Calcule o número de tarefas que possuam altura igual x para cada lista q_i e q_j;
4: **se** $q_i - q_j > \sigma$ **and** $q_i \neq q_j$ **então**
5: Escolha, aleatoriamente, uma tarefa t_k com altura x de l_i.
6: **se** t_k pode ser executada em C_j **então**
7: Insira t_k em l_j, gere c';
8: **fim se**
9: **fim se**
10: **se** $q_i - q_j \leq \sigma$ **então**
11: Escolha, aleatoriamente, duas tarefas t_i e t_j com altura x respectivamente de l_i, l_j.
12: **se** t_i pode ser executada em C_j e t_j pode ser executada em C_i **então**
13: Troque t_i e t_j, gere c';
14: **fim se**
15: **fim se**
16: **retorne:** c';

4.4.3 Codificação de Processadores com DVFS (DVFSE)

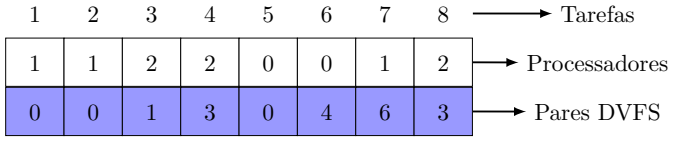
Utilizando a abordagem baseada na alocação de processadores, Guzek et al. (2014) definem a representação dos cromossomos através de dois vetores: (i) o primeiro contempla a alocação de processadores para cada tarefa; (ii) o segundo mapeia qual par DVFS[66] será utilizado pelo processador para executar a tarefa. Para poder avaliar as soluções, Guzek et al. (2014) combinam a estrutura do DAG com uma heurística gulosa como mecanismo externo para a obter o escalonamento final. Em adição, a Codificação de Processadores com DVFS (DVFSE[67]) restringe o conjunto de soluções válidas, consequentemente removendo soluções fracas ou claramente infactíveis. Essas restrições tornam-se uteis à medida que o problema torna-se mais complexo devido à adição do segundo vetor com variáveis de decisão na representação da solução (GUZEK et al., 2014). Todavia, conforme observado pelos autores, a adição dos pares DVFS aumenta o espaço de busca exponencialmente, uma vez que cenários sem o sistema DVFS são representados através de m^n (considerando n tarefas para m máquinas). Cenários com o sistema DVFS habilitado são representados através de $(\sum_{j=0}^{m} pares(j))^n$ ($pares(j)$ contém todos os possíveis pares DVFS associados a uma máquina j).

A Figura 78(a) apresenta um exemplo da DVSFE em um cenário com três processadores heterogêneos, onde os possíveis pares de DVFS que os processadores operam estão

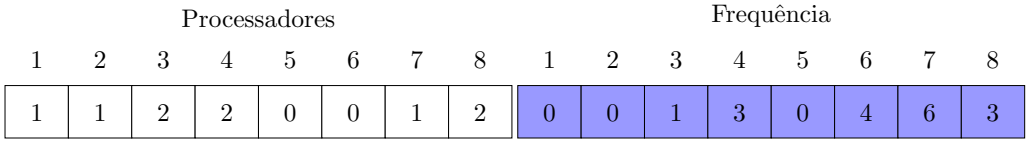
[66] Do inglês, *Dynamic Voltage Frequency Scaling* (Escala de Frequência de Voltagem Dinâmica): tecnologia para controlar frequência de processamento e energia gasta no processo. DVFS é contextualizada na Seção 2.1.2.2
[67] Do inglês, *Dynamic Voltage Frequency Scaling Encoding.*

relacionados na Tabela 4. Além de identificar os genes, a primeira linha do cromossomo também representa todas as tarefas dispostas no grafo de tarefas. A segunda linha apresenta identificadores dos processadores que definem onde as tarefas serão computadas. Por fim, a terceira linha reúne os possíveis pares DVFS aplicados aos processadores, determinando assim o nível de voltagem utilizado. A título de exemplo, a tarefa t_3 está alocada ao processador p_2 que operará sob o par DVFS 3. Outro adaptação da codificação é ilustrada na Figura 78(b), onde Sheikh, Ahmad e Fan (2016) distribuem todas as variáveis de decisão, processadores e frequência, em uma única cadeia do cromossomo.

Figura 78 – Codificação de Processadores com DVFS.



(a) Pares de processadores e DVFS.



(b) Cadeia com processadores e DVFS.

Fonte: adaptado de Guzek et al. (2014), e Sheikh, Ahmad e Fan (2016).

4.4.3.1 Recombinações

As próximas subseções apresentam diversas recombinações empregadas na DVFSE. A primeira subseção contextualiza uma recombinação que efetua trocas de genes baseando-se na formação de um grupo relacionado de tarefas. Uma recombinação baseada no formato uniforme de aplicação é apresentada subseção seguinte. Por fim, a subseção final introduz uma recombinação baseada em distribuições binomiais.

Recombinação por Agrupamento (GX)

A Recombinação por Agrupamento (GX[68]) permite que a operação atue em um grupo de tarefas e não somente em uma única tarefa. O operador também limita-se a não alterar os pares DVFS alocados aos processadores. Ao estar apto a poder combinar grupos inteiros de tarefas, o operador GX acaba trabalhando em um nível mais alto do que recombinações de um ou dois pontos. Seguindo a definição de Guzek et al. (2014), esse

[68] Do inglês, *Grouping Crossover*.

operador tem maior probabilidade que grupos de tarefas sejam fundidos e executados na mesma máquina, o que conduz, consequentemente, a uma redução no tempo de finalização como resultado da diminuição no tempo de comunicação.

Inicialmente, segundo a descrição de Guzek et al. (2014), o operador define aleatoriamente um subconjunto formado pelos processadores utilizados na solução. Esse subconjunto é formado auxiliadamente por uma parte grupal de cada uma das soluções, isto é, uma permutação da lista de processadores utilizada na solução. As instruções utilizadas na geração da parte grupal de cada solução são sumarizadas no Algoritmo 31.

Algoritmo 31 Geração da Parte do grupo – adaptado de Guzek et al. (2014)

entrada: Cromossomo c, número do processador $proc\_num$;
saída: Subconjunto $parte\_grupo$;

```
1:  aux_vetor ← criar_vetor(inteiro, proc_num);
2:  para todo i em 0; obter_tamanho(aux_vetor) − 1 faça
3:     aux_vetor[i] ← −1;
4:  fim para
    processadores ← 0;
5:  para todo i em 0; obter_tamanho(c) − 1 faça
6:     gene ← obter_gene(c, i);
7:     pa ← obter_designação_processador(gene);
8:     se aux_vetor[pa] = −1 então
9:        aux_vetor[pa] ← pa;
10:       processadores ← processadores + 1;
11:    fim se
12: fim para
13: parte_grupo ← criar_vetor(inteiro, processadores);
14: j ← 0;
15: para todo i em 0; obter_tamanho(aux_vetor) − 1 faça
16:    se aux_vetor[i] ≠ −1 então
17:       parte_grupo[j] ← aux_vetor[i];
18:       j ← j + 1;
19:    fim se
20: fim para
21: para todo i em 0; processadores − 1 faça
22:    r ← sortear(i, processadores);
23:    trocar(parte_grupo[i], parte_grupo[r]);
24: fim para
25: retorne: parte_grupo;
```

Em seguida, a parte grupal, obtida no início da recombinação, é dividia sob um ponto de corte definido aleatoriamente. Dessa forma, o subconjunto utilizado para recombinação é formado por todos os processadores dispostos entre o final do cromossomo (extrema direita) e o ponto de corte. Por fim, genes que alocam processadores dispostos no subconjunto são copiados do segundo cromossomo pai ao cromossomo descendente. O Algoritmo 32 apresenta um pseudo-código para a execução completa da recombinação.

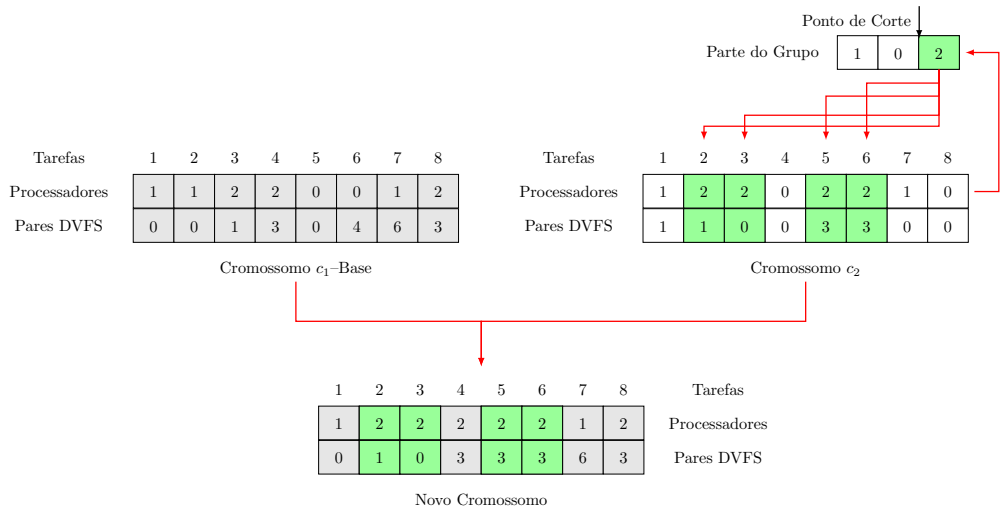
Algoritmo 32 Recombinação por agrupamento – adaptado de Guzek et al. (2014)

entrada: soluções c_1, c_2
saída: cromossomos descendente c'_1

```
1: c'_1 ← c_1;
2: grupo ← gerar_parte_grupo(c_2);
3: g_l ← obter_tamanho(grupo);
4: r ← sortear(0, g_l − 1);
5: para todo i em r; g_l − 1 faça
6:    para todo j em 0; obter_tamanho(c'_1) − 1 faça
7:       tmp_gene ← obter_gene(c_2, j);
8:       se obter_alocacao_processadores(tmp_gene) = grupo[i] então
9:          add_gene(c'_1, j, tmp_gene)
10:      fim se
11:      retorne: c'_1
12:   fim para
13: fim para
```

A Figura 79 ilustra a aplicação do operador GX. Inicialmente, o único cromossomo filho é iniciado através de uma cópia dos genes do cromossomo c_1. Logo após, o subgrupo do operador é formado com base no cromossomo c_2. Aleatoriamente, seleciona-se um ponto de corte que divide uma seção no subgrupo. No exemplo da Figura 79, a divisão gera uma seção no subgrupo possuindo somente um elemento, o processador 2. Ao final, todos os elementos, que estão presente na seção do subgrupo, são copiados ao cromossomo filho em suas respectivas posições. Neste exemplo, os genes presentes nos alelos 2, 3, 5 e 6.

Figura 79 – Recombinação por agrupamento na DVFSE.



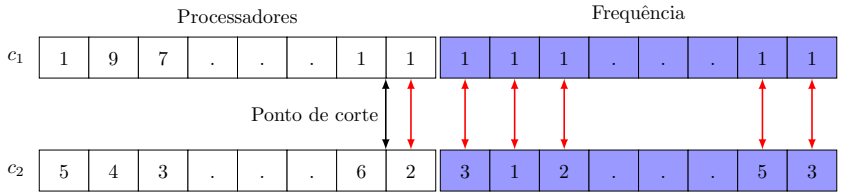
Fonte: adaptado de Guzek et al. (2014).

Recombinação Uniforme (UX)

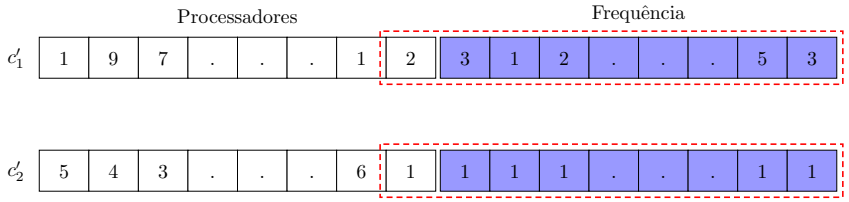
A Recombinação Uniforme (UX) aplicada por Sheikh, Ahmad e Fan (2016) é análoga à recombinação de um ponto. Selecionando-se, aleatoriamente, um ponto de corte no mesmo intervalo dentre um par de cromossomos, efetua-se a troca de todas as variáveis de decisão dispostas após o ponto de corte sorteado. Uma vez que o operador é aplicado à segunda versão do DVFS-E (Figura 78(b)), isto é, um cromossomo unidimensional com intervalo entre $[1, ..., 2n]$, a recombinação pode trocar, simultaneamente, variáveis relacionadas aos processadores e aos níveis de frequência.

A Figura 80 apresenta um exemplo da aplicação do operador uniforme onde dois novos cromossomos são formados com alterações nos processadores e na frequência de operação. Previamente, na Figura 80(a), o ponto de corte sorteado divide os cromossomos c_1 e c_2 em duas partes. Em seguida, na Figura 80(b), os novos cromossomos c'_1 e c'_2 são formados com a troca das variáveis de decisão.

Figura 80 – Recombinação Uniforme na DVFSE.



(a) Cromossomos c_1 e c_2.



(b) Resultado da Recombinação.

Fonte: adaptado de Sheikh, Ahmad e Fan (2016).

Recombinação por Simulação Binária (SBX)

Proposto ao MSP por Sheikh, Ahmad e Fan (2016) como uma operação adicional à recombinação uniforme (Subseção 4.4.3.1), a Recombinação por Simulação Binária (SBX[69]) trabalha com pares de indivíduos e com a geração de duas distribuições binomiais. SBX é melhor adaptável em problemas com variáveis de decisão não binárias (DEB; AGRAWAL, 1994). Além disso, o uso das distribuições permite que os cromossomos descendentes possuam valores perto de seus pais, quando a diferença entre os pais é pequena, e ao mesmo

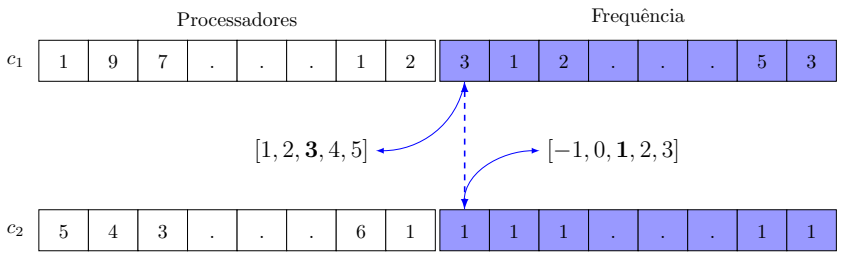
[69] Do inglês, *Simulated Binary Crossover*.

tempo, permite que os descendentes diferenciem-se significativamente, quando as diferença entre os pais é grande (DEB, 2001).

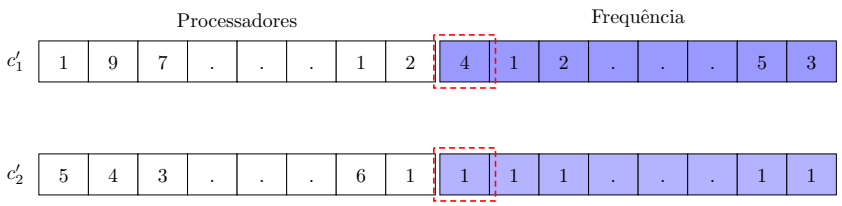
A geração das distribuições obedece a distribuição binomial $B(pai\_val, 2\Delta + 1)$, onde pai_val é o valor da variável de decisão de um cromossomo pai e Δ é a diferença entre as variáveis de decisão de ambos os pais. Consequentemente, no trabalho de Sheikh, Ahmad e Fan (2016), a expressão $B(\alpha, \beta)$ representa uma distribuição binomial de tamanho β com α ao centro. Desde que o SBX é aplicado para cada variável de decisão individualmente, pode haver duas probabilidades associadas. Sheikh, Ahmad e Fan (2016) definem as probabilidades supracitadas como $p_{c_{mem}}$, a chance de um par participar da recombinação, e $p_{c_{dec}}$, a chance de uma variável de decisão participar de uma recombinação.

A Figura 81 ilustra um exemplo da recombinação por simulação binária, onde a aplicação é efetuada nos cromossomos c_1 e c_2, respectivamente, às variáveis de decisão com valores 3 e 1. Uma distribuição binomial é formada para cada um dos pais. Inicialmente, a primeira distribuição, gerada em c_1, possui o valor 3 em seu centro, enquanto que a segunda distribuição, gerada em c_2, possui o valor 1. O tamanho de ambas as distribuições é 5 uma vez que $\Delta = 2$. Desta forma, a primeira distribuição é formada por $B(3, 5)$, e a segunda distribuição é formada por $B(1, 5)$. Conforme apontado por Sheikh, Ahmad e Fan (2016), se os valores gerados pela distribuição excederem os limites inferiores ou superiores das variáveis de decisão, o novo valor da variável será fixado para o valor mais próximo do limite inferior (quando menor) ou superior (quando maior).

Figura 81 – Recombinação por Simulação Binária.



(a) Cromossomos c_1 e c_2.



(b) Resultado da Recombinação.

Fonte: adaptado de Sheikh, Ahmad e Fan (2016).

4.4.3.2 Mutações

Com a combinação entre alocação e pares DVFS, a DVFSE apresenta mutações voltadas a modificação dessa união. A primeira subseção apresenta uma mutação com variações sobre o local de aplicação. Ao final, a outra mutação apresentada funciona sobre distribuições uniformes de valores.

Mutação por Troca de DVFS (DVFSM)

O trabalho de Guzek et al. (2014) apresenta duas variações de Mutação por Troca de DVFS (DVFSM[70]), Tipo 1 e Tipo 2, aplicadas ao mapeamento de pares DVFS de cada cromossomo. Ambos os operadores possuem funcionamento baseado em sorteios das alocações, comumente resumido como operadores *bit-flip*.

Na mutação de Tipo 1, um gene terá seu processador e o respectivo par DVFS realocados. Inicialmente, seleciona-se aleatoriamente um dos genes dispostos no cromossomo. Em seguida, dentre a lista de possíveis processadores no cenário do problema, um novo processador será sorteado e alocado ao gene selecionado. Por fim, considerando a heterogeneidade de cada processador, um novo par DVFS será selecionado dentre os possíveis pares disponíveis no novo processador.

As operações executadas na mutação de Tipo 2 são análogas à mutação de Tipo 1, exceto que a modificação é exclusivamente aplicada ao par DVFS. Dessa forma, aleatoriamente, seleciona-se um gene, o qual terá seu par DVFS modificado. Conforme o operador anterior, o sorteio do novo par DVFS é limitado aos possíveis pares dispostos no processador alocado.

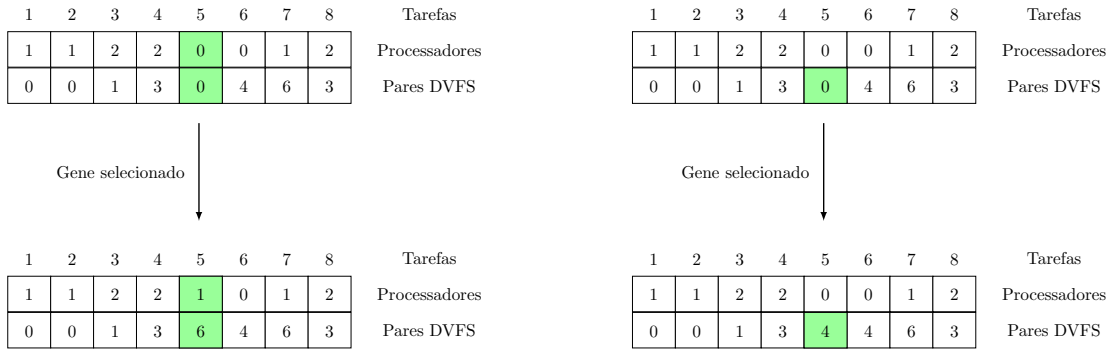
A Figura 82 apresenta um exemplos de ambos os operadores. Na Figura 82(a) a mutação de Tipo 1 é aplicada ao quinto gene do cromossomo, respectivamente, a tarefa t_5 tem seu processador e par DVFS alterados aleatoriamente. Em contrapartida, a Figura 82(b) apresenta a aplicação da mutação de Tipo 2 também à tarefa t_5, todavia, com uma alteração menor comparada ao Tipo 1.

Mutação de Distribuição Uniforme (UDM)

A Mutação de Distribuição Uniforme (UDM[71]) é caracterizada pela geração de duas distribuições uniformes, com intervalos em $[v\_inferior, valor]$ e $[valor, v\_superior]$, para cada variável de decisão. As variáveis $v\_inferior$ e $v\_superior$ representam os limites inferiores e superiores para cada variável de decisão. Conjuntamente, os limites inferiores e superiores para as variáveis de decisão voltadas à alocação de processadores e à definição da frequência são, respectivamente, $(1, M)$ e $(1, K)$.

[70] Do inglês, *Dynamic Voltage Frequency Scaling Mutation*.
[71] Do inglês, *Uniform Distributation Mutation*.

Figura 82 – Mutação por Troca de DVFS.

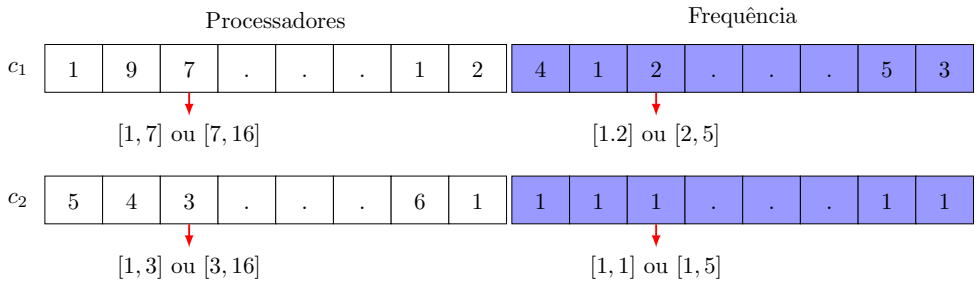


(a) Mutação de Tipo 1. (b) Mutação de Tipo 2.

Fonte: adaptado de Guzek et al. (2014).

A Figura 83 demonstra um exemplo da geração das destruições aplicadas em dois cromossomos diferentes, utilizando um cenário homogêneo com $M = 16$ e $K = 5$. Ambos os cromossomos adotam a segunda representação empregada em pares DVFS, conforme a Figura 78(b). De acordo com a definição de Sheikh, Ahmad e Fan (2016), após a geração das distribuições, para aplicar o operador de mutação, aleatoriamente, seleciona-se uma das distribuições para em seguida, utilizá-la para gerar o valor da variável de decisão.

Figura 83 – Mutação de Distribuição Uniforme.



Fonte: adaptado de Sheikh, Ahmad e Fan (2016).

4.4.4 Codificação de Q-bits (QE)

Finalmente, o trabalho de Gandhi, Nitin e Alam (2017) apresenta um Algoritmo Genético Quântico com refinamento de rotação de ângulo. A diferença principal desse modelo é a codificação empregada, sendo essa formada por uma coleção de qubits. Na computação quântica, um qubit (ou bit quântico) é a menor unidade de armazenamento de informação. Um qubit pode estar no estado 0, no estado 1 ou em uma sobreposição

de ambos (GANDHI; NITIN; ALAM, 2017). A Expressão 31 representa o estado de um qubit.

$$|\Psi >= \alpha|0 > +\beta|1 > \tag{31}$$

Os termos α e β são números complexos que obedecem a restrição da Expressão 32:

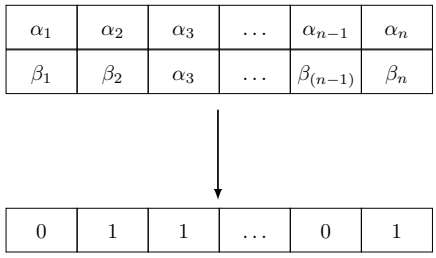
$$|\alpha|^2 + |\beta|^2 = 1 \tag{32}$$

No trabalho de Gandhi, Nitin e Alam (2017), β representa a probabilidade do qubit estar no estado 1, enquanto que α representa a probabilidade de um qubit estar no estado 0, ou vice-versa. Uma coleção de k qubits pode representar 2^k possíveis estados. Algoritmos quânticos podem utilizar várias portas quânticas: porta Not, porta C-Not, porta de rotação, dentre outras. Gandhi, Nitin e Alam (2017) define um método de refinamento da população usando uma porta de rotação (Expressão 33), sendo esta definida pela magnitude e ângulo de rotação θ.

$$\begin{bmatrix} \alpha_i^{t+1} \\ \beta_i^{t+1} \end{bmatrix} = \begin{bmatrix} cos(\Delta\theta_i) - sin(\Delta\theta_i) \\ sin(\Delta\theta_i) \quad cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i^t \\ \beta_i^t \end{bmatrix} \tag{33}$$

Na Codificação de Q-bits (QE[72]), Gandhi, Nitin e Alam (2017) formam um cromossomo como uma coleção de qubits. Uma vez que estes qubits podem representar uma sobreposição de todos os valores possíveis, a propriedade explorativa da abordagem genética sofre um aumento. Para o cálculo da aptidão, é necessário a decodificação de uma nova população. A Figura 84 ilustra a extração de um cromossomo clássico demonstrando a conversão de uma população quântica para uma população binária. O processo é executado através da geração de números aleatórios pela checagem do valor de α^2 contra um número aleatório. A população binária é mapeada para valores inteiros correspondentes a contagem binária de processador.

Figura 84 – Decodificando um cromossomo quântico para um cromossomo binário.



Fonte: adaptado de Gandhi, Nitin e Alam (2017).

[72] Do inglês, *Q-bit Encoding*.

O operador SPX é empregado nas recombinações executadas na QE. O funcionamento é análogo às codificações supracitadas com a única diferença caracterizada pela aplicação ser voltada ao vetor de qubits. Não apresentando um método de mutação, a QE efetua uma combinação de geração de ângulo de rotação juntamente à porta de rotação para um refinamento quântico.

4.5 Sumário

Além de levantar uma coleção de estudos, a SLR foi executada com a finalidade de extrair as principais características dos GAs implementados nesses trabalhos: codificações de indivíduos, recombinações, mutações e funções objetivo. Desse modo, esta seção sintetiza as principais informações referentes às codificações e aos operadores desse capítulo. Além disso, essa seção complementa a resposta à primeira questão secundária de pesquisa (**QS.01**).

As representações variam suas características devido às métricas empregadas ou por outros fatores que influenciem a complexidade. Apesar disso, Guzek et al. (2014) dividem essas representações em três grupos principais:

- **Ordem de nós:** lida somente com a prioridade das tarefas;
- **Alocação de processadores:** controla a designação de tarefas aos processadores disponíveis no cenário;
- **Representação direta:** resulta em um exato mapeamento ao combinar a alocação com as restrições de prioridade, todavia, apresenta operadores genéticos complexos e um aumento árduo na complexidade.

Guzek et al. (2014) ainda complementam que os grupos de representação baseada em listas de nós e alocação de processadores devem ser combinados com algum método externo para a criação do escalonamento final. Baseada na classificação dos autores, a Tabela 8 distribui as codificações levantadas no capítulo. Além disso, a tabela expande a divisão dos autores ao categorizar as representações específicas de certos cenários. Dentre todas as codificações, a representação direta (alocação e ordenação) é o cenário mais explorado.

Tabela 8 – Classificações de codificações.

Ordem	**Alocação**	**Alocação e Ordenação**	**Outras**
OLE, PE	PLE, FTE, RE	TLE, ME, MSE, TE	MLE, DLE, DVFSE, QE

A Tabela 9 sumariza cronologicamente os trabalhos pesquisados e seus algoritmos, juntamente com os cenários de aplicação, sendo CC, HOMO e HETE as abreviações para Custo de Comunicação, Homogêneo e Heterogêneo, respectivamente. Os trabalhos

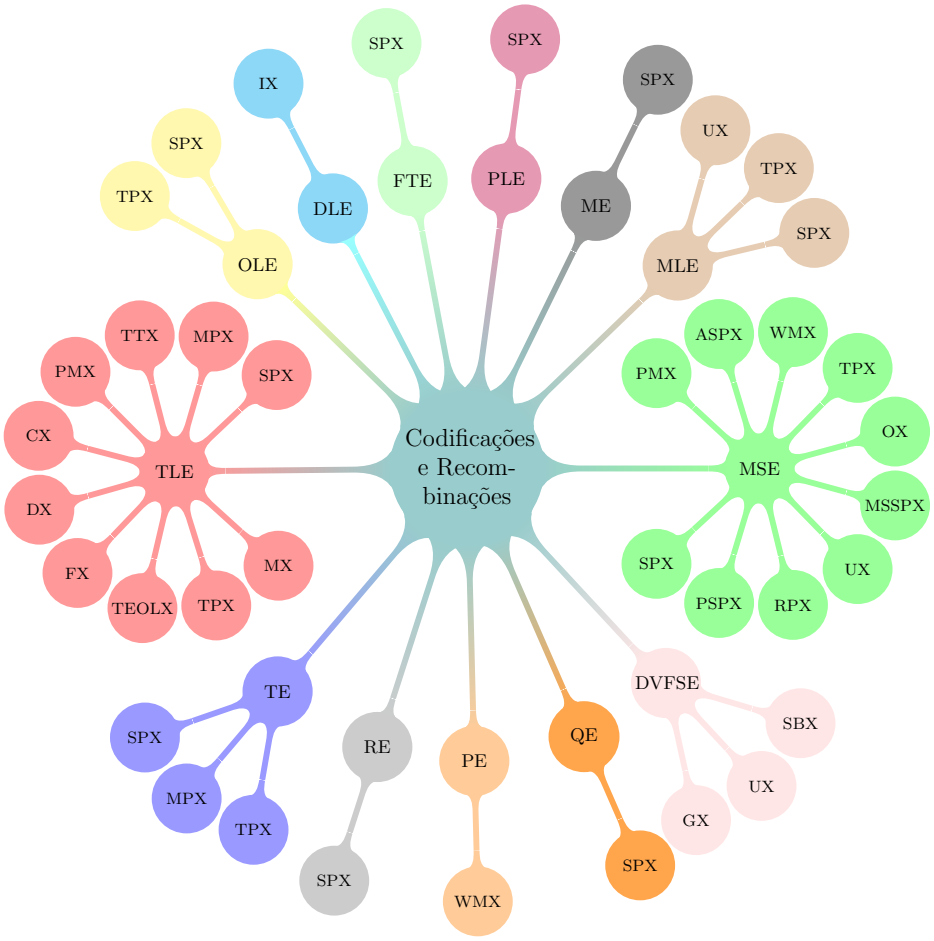
são referenciados pelos IDs presentes na Tabela 5. Células com um hífen identificam obras em que os autores não explicitaram a técnica ou cenário utilizados. Um trabalho ainda pode apresentar mais de uma codificação ou operador, por exemplo, o trabalho de Omara e Arafa (2009) emprega a Codificação de Alocação e Escalonamento (MSE) e a Codificação de Tuplas (TE) com diversas variações. Os dados coletados demonstram uma grande variedade de técnicas voltadas a potencializar o desempenho de soluções ao MTSP. Adicionalmente, também é possível observar uma flexibilidade de mecanismos voltados a diversos cenários e limitações.

Ao relacionar os dados da Tabela 5, a Figura 85 sintetiza uma distribuição hierárquica entre as codificações levantadas e seus respectivos operadores de recombinação. Naturalmente, a Codificação de Alocação e Escalonamento (MSE) e a Codificação de Listas Topológicas (TLE) são as representações com a maior variedade de recombinações. Além disso, a distribuição demonstra a popularidade do operador de Recombinação de Ponto Único (SPX) na maior parte das codificações. Entretanto, cabe ressaltar que um mesmo operador aplicado em diversas codificações possui mecanismos exclusivos para cada modelo, por exemplo, o SPX aplicado na TLE lida com restrições de precedência, enquanto que na Codificação de Lista de Processadores (PLE), o mecanismo é aplicado sem maiores restrições.

Uma outra sintetização é ilustrada na Figura 86, desta vez, distribuindo as codificações e os operadores de mutação levantados. Semelhante à análise anterior, MSE e TLE são as codificações com a maior variedade de mutações. Além disso, os operadores de mutação também são amplamente relacionados à codificação utilizada, de forma que uma mutação por troca de tarefas não pode ser empregada em uma representação exclusivamente de alocação.

Uma relação entre o período de publicação analisado e o surgimento das codificações levantadas é ilustrada na Figura 87. Com a utilização de listas representando cada processador, a Codificação de Listas Topológicas (TLE) é o primeiro modelo adotado, sendo introduzido inicialmente por Hou, Hong e Ansari (1990). Em seguida, Benten e Sait (1994) empregam a codificação por Lista de Processadores (PLE) relacionando exclusivamente os processadores do sistema. Quase no mesmo período, um modelo baseado em matrizes, Codificação por Matrizes (ME), é apresentado por Wang e Korfhage (1995). A próxima representação introduzida foi o mapeamento gerado pela Codificação por Lista de Mapeamentos (MLE), sendo utilizada por Singh e Youssef (1996). No mesmo ano, Wang, Siegel e Roychowdhury (1996) relacionam ordenação e alocação na formação da Codificação de Alocação e Escalonamento (MSE). Posteriormente, uma representação empregando exclusivamente a ordenação, Codificação por Lista Ordenada (OLE), é apresentada por Kwok e Ahmad (1997). Um tempo depois, o modelo baseado em tuplas, Codificação por Tuplas (TE), e o modelo de dois níveis, Codificação de Duplo Nível (DLE), são apresentados, sendo estes representados por Topcuoglu e Sevilmis (2002) e Liu, Li e Yu (2002),

Figura 85 – Distribuição hierárquica entre codificações e recombinações.

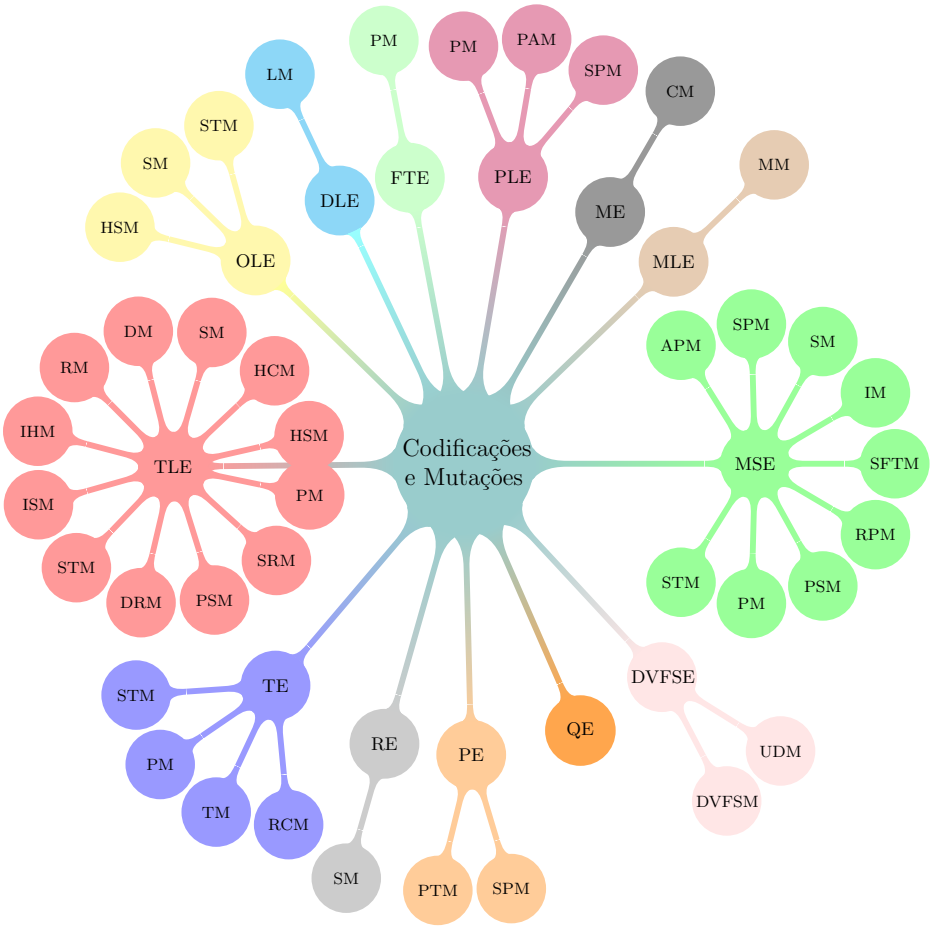


Fonte: do autor.

respectivamente. A próxima representação foi baseada em um modelo fixo de tarefas, Codificação de Tarefa Fixa (FTE), e foi apresentado por Jelodar et al. (2006). Utilizando índices de prioridade, Hwang, Gen e Katayama (2008) apresentaram a Codificação de Prioridade (PE). A Codificação de Processadores com DVFS (DVFSE), empregada por Guzek et al. (2014), reúne conceitos de alocação, ordenação e sustentabilidade energética. Experimentando diversas codificações, Hassan et al. (2015) propuseram a Codificação de Intervalo (RE) em um de seus GAs. Por fim, Gandhi, Nitin e Alam (2017) montaram a Codificação de Q-bits (QE) para o tratamento do problema com recursos de computação quântica.

Adicionalmente, a Figura 88 quantifica a aplicação das codificações levantadas. A MSE é a representação mais utilizada, estando presente em 19 aplicações ($\approx 27.94\%$). A segunda representação mais utilizada é a TLE com 17 aplicações ($\approx 25\%$). A TLE é a codificação pioneira encontrada na coleção de trabalhos desse estudo, estando presente inicialmente no trabalho de Hou, Hong e Ansari (1990). Codificações rotuladas como outras referem-se aos modelos com um única aplicação dentre os trabalhos da coleção. A

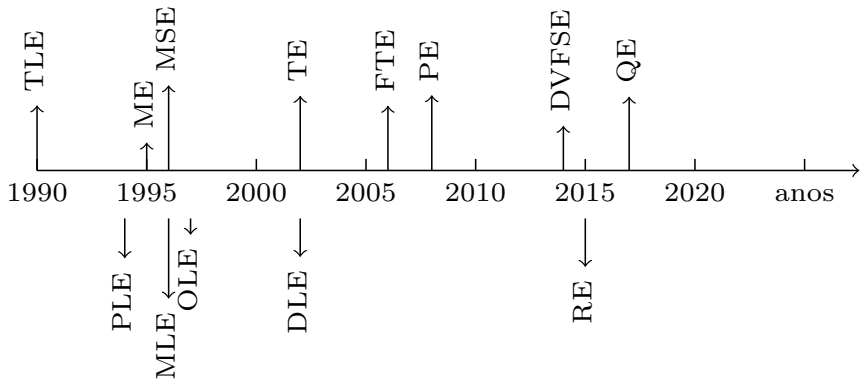
Figura 86 – Distribuição hierárquica entre codificações e mutações.



Fonte: do autor.

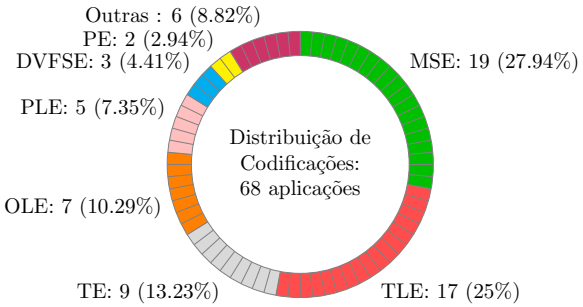
utilização de diversas codificações no MTSP revela o esforço dos pesquisadores em encontrar representações adequadas aos cenários e que possam oferecer melhores resultados de desempenho.

Figura 87 – Linha do tempo com as codificações levantadas por data.



Fonte: do autor.

Figura 88 – Contabilização de aplicações das codificações.



Fonte: do autor.

Tabela 9 – Características extraídas dos artigos.

ID	Ano	Codificação	Recombinação	Mutação	CC	Ambiente
1	1990	TLE	SPX	-	Não	HOMO
2	1994	PLE	SPX	SPM	Sim	HOMO
3	1994	TLE	SPX	HSM	Não	HOMO
4	1995	ME	SPX	CM	Sim	HOMO
5	1996	TLE	-	IHM;HCM	Não	HOMO
6	1996	MLE	SPX; TPX; UX	MM	Sim	HETE
7	1996	MSE	SPX	STM; PM	Sim	HETE
8	1997	OLE	SPX	STM	Sim	HOMO
9	1997	TLE	MPX	SM; DM; RM	Sim	HOMO
10	1997	PLE	-	-	Sim	HOMO
11	1997	TLE	TTX	ISM	Sim	HOMO/HETE
12	1997	MSE	SPX	STM; PM	Sim	HETE
13	1998	TLE	MPX	SM; DM; RM	Sim	HOMO
14	1999	TLE	SPX	STM	Sim	HOMO
15	1999	TLE	PMX; CX	HSM; PM	Não	HOMO
16	1999	TLE	DX	DRM	Sim	HOMO
17	2002	TLE	FX	PSM	Não	HOMO
18	2002	DLE	IX	LM	Não	HETE
19	2002	TE	SPX	STM; PM	Sim	HETE
20	2003	MSE	PSPX; RPX	PSM; PM	Sim	HOMO
21	2003	TLE	TEOLX	SRM	Sim	HOMO
22	2004	MSE	UX; MSSPX	RPM; PM	Sim	HOMO
23	2004	TE	MPX	TM; PM	Sim	HOMO/HETE
24	2006	FTE	SPX	PM	Sim	HOMO/HETE
25	2006	TE	SPX	STM; PM	Sim	HETE
26	2006	OLE	SPX	STM	Não	HOMO
27	2008	PE	WMX	PTM	Sim	HOMO
28	2008	PE	WMX	SPM	Sim	-
29	2009	MSE; TE	SPX; OX; TPX	PM	Sim	HOMO
30	2009	TE	-	-	Sim	HETE
31	2009	MSE	WMX; SPX; OX	SM; SFTM; IM	Sim	HOMO
32	2010	TLE	SPX; TPX	HSM; PSM; PM	Sim	HOMO
33	2010	MSE	TPX	STM; PM	Não	HETE
34	2010	MSE	SPX; TPX	PM; SM	Sim	HETE
35	2010	TLE	SPX; TPX	HSM; PM; PSM	Sim	HOMO
36	2011	MSE	SPX	SM; SPM; PM; PSM	Sim	HOMO
37	2011	PLE	SPX	PAM	Não	HETE
38	2011	MSE	ASPX	APM	Sim	HETE
39	2011	MSE	SPX	STM; PM	Sim	HETE
40	2011	MSE	SPX	STM; PM	Sim	HETE
41	2011	TLE	MX	HSM	Sim	HETE
42	2012	MSE	SPX; TPX	PM	Sim	HETE
43	2012	MSE	SPX	PM	Sim	HOMO
44	2012	TE	SPX	RCM	Não	HETE
45	2012	OLE	SPX	STM	Sim	HETE
46	2012	TLE	SPX; TPX	SM	Sim	HETE
47	2012	TLE	SPX; TPX	PM	Sim	HOMO/HETE
48	2012	MSE	-	SM; PSM	Sim	HOMO/HETE
49	2013	OLE; PLE; MSE	SPX	SM; SPM; PM	Sim	HOMO
50	2014	OLE	SPX	STM	Sim	HETE
51	2014	MSE	SPX; OX; PMX; PX	PM; SM; SFTM; IM	Sim	HOMO
52	2014	OLE	SPX; TPX	HSM	Não	HOMO
53	2014	DVFSE	GX	DVFSM	Sim	HETE
54	2015	OLE; RE; PLE	SPX	SM; PM	Não	HETE
55	2016	DVFSE	UX; SBX	UDM	Sim	HOMO
56	2016	MSE	PMX; TPX	SM; PM	Sim	HOMO
57	2016	MSE	SPX; TPX	PM	Sim	HETE
58	2017	TE	SPX	STM	Sim	HETE
59	2017	TE	TPX	PM	Sim	HOMO
60	2017	QE	SPX	-	Sim	HETE
61	2017	DVFSE	-	-	Sim	HOMO
62	2018	TE	SPX	STM	Sim	HETE
63	2018	TLE	SPX	HSM	Não	HOMO

Capítulo 5

Funções Objetivo

Este capítulo sumariza as funções objetivo levantadas na coleção e as distribui sobre as codificações identificadas. Dessa forma, as seções deste capítulo contribuem com a resposta da quinta questão secundária de pesquisa (**QS.05**), apresentada no Capítulo 3. Além disso, alguns procedimentos necessários no cálculo dos resultados são contextualizados e referenciados. Ao contrário do capítulo anterior, a distribuição das seções deste capítulo representa uma ordem cronológica das representações levantadas. Entretanto, ao final do capítulo, uma seção de sumário agrupa as funções objetivo e as representações nas categorias apresentadas no Capítulo 4.

Inicialmente, a Seção 5.1 apresenta funções objetivo sobre a representação pioneira TLE. Em seguida, as funções objetivo designadas à PLE são apresentadas na Seção 5.2. A Seção 5.3 aborda a avaliação efetuada na representação MLE. As funções objetivo levantadas na MSE são apresentadas na Seção 5.4. A Seção 5.5 lista as funções empregadas na representação OLE. As funções objetivo da representação TE são apresentadas na Seção 5.6. A Seção 5.7 contextualiza as funções objetivo das representações DLE, FTE e PE. As funções objetivo e os mecanismos necessários a avaliação na representação DVFSE são contextualizadas na Seção 5.8. A Seção 5.9 lista as codificações em que os estudos não apresentam as funções objetivo explicitamente. Por fim, a Seção 5.10 apresenta uma síntese das funções objetivo levantadas na coleção juntamente às suas respectivas codificações.

5.1 Função Objetivo na TLE

O tempo de finalização (*makespan*) de um escalonamento é a métrica mais comum utilizada na avaliação de desempenho de um algoritmo. A princípio, Hou, Hong e Ansari (1990) definiram um método para cálculo do tempo de finalização, apresentado no Algoritmo 33 e construído sobre três funções essenciais, sendo essas:

- $et(t_i)$: tempo de execução de uma tarefa t_i;

- ❏ $ftt(t_i)$: tempo de finalização da tarefa t_i;
- ❏ $ftp(p_j)$: tempo de finalização do processador p_j.

Algoritmo 33 Cálculo para tempo de finalização total – adaptado de Hou, Hong e Ansari (1990)

entrada: Escalonamento S;
saída: Tempo de finalização FT;
{Inicialização}
1: Para cada processador p_j, $ftp(p_j) \leftarrow 0$;
{Laço para cada tarefa t_i}
2: Para cada tarefa t_i no processador p_j, repita os passos 3, 4 e 5;
{Tarefa t_i não tem predecessores}
3: Se $pred(t_i) = \emptyset$, $ftt(t_i) \leftarrow et(t_i) + ftp(p_j)$
{Tarefa t_i tem predecessores}
4: Caso contrário, $ftt(t_i) \leftarrow et(t_i) + \max ftt(t_k), ftp(p_j)$ onde $t_k \in pred(t_j)$
{Atualizar o tempo de finalização do processador p_j}
5: $ftp(p_j) \leftarrow ftp(p_j) + ftt(t_i)$
{Calcular o tempo de finalização do escalonamento}
6: $FT \leftarrow \max ftp(p_j)$ sobre todo p_j

Outros autores simplificam a função objetivo voltada ao tempo máximo de finalização da última tarefa, como, por exemplo, Woo et al. (1997). Considerando uma função de ativação, f, aplicada em um cromossomo c, a Expressão 34 a seguir é aplicada:

$$\begin{aligned} f &= \max_{i \leq u \leq m} ftp(p_u), \\ \text{onde } ftp(p_u) &= \max_{t_i \in PT} T(t_i, p_u) \text{ e} \\ PT &\text{ é o conjunto de tarefas assinaladas para } p_u \end{aligned} \tag{34}$$

O valor de f representa o tempo total necessário para a execução de todas as tarefas seguindo o escalonamento codificado pelo cromossomo. Na métrica do tempo de execução, quanto menor o valor final, melhor será o escalonamento. Esse, portanto, é um problema de minimização. Todavia, para o funcionamento de algumas operações de reprodução, pode-se adaptar os resultados de forma a maximizar os resultados, consequentemente convertendo o problema para uma maximização.

Hou, Hong e Ansari (1990) e Tsujimura e Gen (1996) obtêm a maximização através de $1/FT$, caracterizando uma estratégia por divisão. Em contrapartida, considerando FT_{max} como o tempo de finalização máximo, Hou, Ansari e Ren (1994) propõem uma maximização como $FT_{max} - FT(s)$, onde s é um escalonamento produzido por um cromossomo c.

Em um ambiente com TD, Tsuchiya, Osada e Kikuno (1997) obtêm a maximização da função objetivo com a Expressão 35, onde $comprimento(s)$ corresponde ao comprimento de um escalonamento s, e ℓ_{max} identifica o comprimento máximo encontrado na população.

$$f = (\ell_{max} + 1 - comprimento(s))^2 \tag{35}$$

Posteriormente, Tsuchiya, Osada e Kikuno (1998) remodelam a função também obtendo a maximização, conforme demonstrado na Expressão 36.

$$f = \left(\frac{(\ell_{max} - comprimento(s))}{10} + 1\right)^2 \tag{36}$$

De forma análoga a esses trabalhos, Corrêa, Ferreira e Rebreyend (1999) definem a aptidão de um cromossomo, em seus dois GAs propostos, como uma diferença entre o seu valor de makespan e o valor de makespan mais alto encontrado na população atual. Portanto, o melhor cromossomo encontrado é aquele com o menor valor de makespan (CORRÊA; FERREIRA; REBREYEND, 1999).

No trabalho de Zomaya, Ward e Macey (1999), após o cálculo do comprimento total do escalonamento do cromossomo, o valor de aptidão é computado conforme a Expressão 37, sendo $FT(c)$ o tempo de finalização do cromossomo c e FT_{max} é o tempo de finalização máximo dentre todos os outros da população. O valor de 1% é acrescentado a todas as aptidões para evitar que, caso um cromossomo obtenha o tempo de finalização máximo, o valor de aptidão seja zerado.

$$f = (FT_{max} - FT(c)) + 0.001 \times FT_{max} \tag{37}$$

Os trabalhos de Jezic et al. (1999) e Singh e Pillai (2014) modelam a maximização através da subtração entre $FT(s)$ e uma grandeza ET que representa os custos de finalização de todas as tarefas ($ET = \sum et(t_i)$). Desse modo, a expressão para aptidão assume o formato $f = ET - FT(s)$.

Em Zhong e Yang (2003), há uma combinação entre os tempos de finalização máximo e mínimo. Em adição, a distribuição de indivíduos pode ser ajustada por um parâmetro r, conforme a expressão:

$$f = \frac{FT_{max} - FT(s) + r}{FT_{max} - FT_{min} + r} \tag{38}$$

As métricas de makespan e flowtime são utilizadas na otimização de Kaur, Chhabra e Singh (2010a) e Kaur, Chhabra e Singh (2010b), sendo ambas calculadas e avaliadas separadamente.

O trabalho de Daoud e Kharma (2011) normaliza a aptidão de um cromossomo como $1/comprimento(s)$, onde, conforme supracitado, $comprimento(s)$ é comprimento de um escalonamento s decodificado.

Para calcular a aptidão de um cromossomo, Singh (2012) e Singh e Singh (2012) combinam duas funções de avaliação: aptidão das tarefas e aptidão dos processadores. Na primeira, pares de tarefas são analisados na busca por violações das restrições de precedência. Por outro lado, a aptidão dos processadores reflete a tentativa de minimizar o tempo de finalização de cada processador.

O trabalho de Pillai et al. (2018) propõe uma otimização multiobjetivo entre as métricas de energia e makespan. Conforme a definição de Pillai et al. (2018), o consumo elétrico de um sistema pode ser calculado conforme a Expressão 39, onde SP é a capacidade de troca, V_{dd} é a voltagem operacional, fo_i é frequência de operação para uma tarefa i, et_i é o tempo de execução de uma tarefa i, e m é a quantidade de tarefas no escalonamento.

$$E = SP \times V_{dd}^2 \times \sum_{i=1}^{m} fo_i \times et_i \tag{39}$$

Em seguida, o consumo elétrico é adaptado à maximização conforme a Expressão 40, sendo E_{max} o consumo elétrico máximo que pode ser demandando e E_i o consumo elétrico de um cromossomo i.

$$f_1 = E_{max} - E_i \tag{40}$$

Da mesma forma que a métrica anterior, o makespan é normalizado para maximização conforme a Expressão 41, onde T_{max} representa o tempo de execução total de todas as tarefas, enquanto que FT_i é o comprimento do escalonamento de um cromossomo i.

$$f_2 = ET - FT_i \tag{41}$$

Por fim, Pillai et al. (2018) implementam a otimização multiobjetivo ao combinar as duas funções de aptidão em uma soma ponderada conforme a Expressão 42, sendo f_1' e f_2' versões normalizadas das funções das Expressões 40 e 41; w_1 e w_2 pesos definidos pelo usuário na forma que $w_1 + w_2 = 1$.

$$f = w_1 f_1' + w_2 f_2' \tag{42}$$

5.2 Função Objetivo na PLE

O trabalho de Benten e Sait (1994) define o valor de aptidão como a máxima soma de tempo requirido para executar todas as tarefas alocadas ao processadores, acrescentando o tempo de ociosidade de cada unidade. A Expressão 43 demonstra o procedimento adotado em Benten e Sait (1994). Em adição, para definir o tempo de conclusão, Benten e Sait (1994) utiliza um esquema com a inserção de nós simulados, de forma que nenhuma tarefa fique no mesmo nível que sua tarefa precedente. Os nós simulados possuem tempo

de execução zerado e são alocados a unidades nulas. A cada nível, o tempo de conclusão é determinado sob um escalonamento parcial.

$$o^{-1} = \max_{j=0,m-1}(\sum_{t_i \in p_j} et(t_i) + ocioso(j)) \tag{43}$$

O trabalho de Aguilar e Gelenbe (1997) utiliza uma função de custo baseando-se no custo do balanceamento de carga, Expressão 45, combinado ao custo de comunicação das tarefas, Expressão 44. Dessa forma, Aguilar e Gelenbe (1997) definem essas métricas como:

- **Custo de comunicação:** a quantidade de dados trocados entre as tarefas, sobre a topologia de conexão. A Expressão 44 obtém o custo de comunicação, onde CCP_{piqj} é o tempo necessário para um processador p estabelecer uma comunicação entre as tarefas i e j, caso i esteja em p e j esteja em q, CI_{ij} é quantidade total de informação trocada entre as tarefas i e j, D_{pq} é o tempo médio necessário para transferir uma informação entre os processadores p e q e X_{ip} é uma variável binária que define uma alocação, ou seja, se a tarefa i estiver alocada no processador p, X_{ip} assume o valor 1 (0 caso contrário) (AGUILAR; GELENBE, 1997).

$$CC_c = \sum_{p}\sum_{q \neq p}\sum_{i}\sum_{j \neq i}(CCP_{piqj} + CCP_{qjpi} + CI_{ij}D_{pq} + CI_{ji}D_{qp})X_{ip}X_{jq} \tag{44}$$

- **O custo do balanceamento de carga:** por fim, uma equilibrada distribuição da carga de trabalho entre os processadores é efetuada. Aguilar e Gelenbe (1997) avaliam esse custo conforme a Expressão 45, onde C_E é o custo de execução total do grafo de tarefas (uma abordagem sequencial do grafo de tarefas ou a quantidade total de trabalho que ele contém), e_i é a quantidade de instruções de uma tarefa, M é a quantidade processadores dispostos no sistema e U_p é o tempo médio de execução de uma instrução em um processador p.

$$C_B = \frac{1}{M}\left(\sum_{p}\left(\sum_{i} e_i U_p X_{ip} - \frac{C_E}{M}\right)^2\right) \tag{45}$$

O primeiro passo para avaliar uma solução no trabalho de Wang et al. (2011) é a definição da ordem de execução representada pelo cromossomo. Neste cenário, Wang et al. (2011) propõem a união de duas heurísticas de prioridade para obter uma ordem de execução otimizada, de forma que essa junção forma uma estratégia de escalonamento max-min. O método assinala alta prioridade às tarefas que podem ser iniciadas primeiro ou que possam influenciar o makespan do programa.

A primeira heurística utiliza o tempo médio de execução de instruções de todas os recursos para estimar o tempo de conclusão do mais longo caminho começando da tarefa. Sendo assim, em um cenário onde duas tarefas estejam escalonadas para o mesmo

processador, a tarefa com a mais alta prioridade deve ser escalonada primeiro (WANG et al., 2011). Inicialmente, a Expressão 46 define a importância $imprt(t_i)$ de uma tarefa t_i como o mais longo caminho iniciando da tarefa no grafo. Em seguida, a Expressão 47 define a prioridade $pri(i)$ de uma tarefa t_i, onde $PE(\gamma)$ é velocidade média de execução de instruções em todos os recursos, γ_i representa o tempo de execução de instrução de um processador p_i, t_i^{avai} indica o tempo de iniciação disponível e $e(i,j)$ representa uma dependência entre as tarefas t_i e t_j.

$$imprt(i) = \begin{cases} |t_i| \text{ se } t_i \text{ é uma tarefa de saída} \\ |t_i| + \max_{e(i,j)\in E} imprt(t_j) \text{ caso contrário} \end{cases} \tag{46}$$

$$pri(i) = PE(\gamma).imprt(t_i) - \max(t_i^{\text{avail}}, ocioso(M(t_i))) \tag{47}$$

De forma análoga, a segunda heurística estima o tempo de conclusão $comp(t_i)$, do mais longo caminho iniciando da tarefa t_i, conforme a Expressão 48, onde $M(i) = r_j$. Desse modo, a Expressão 49 define a prioridade de uma tarefa t_i.

$$comp(i) = \begin{cases} |t_i|.\gamma_j \text{ se } t_i \text{ é uma tarefa de saída} \\ |t_i|.\gamma_j + \max_{e(i,j)\in E} comp(k) \text{ caso contrário} \end{cases} \tag{48}$$

$$pri(i) = comp(i) - \max(t_i^{\text{avail}}, idle(M(i))) \tag{49}$$

Por fim, Wang et al. (2011) combinam as heurísticas de prioridade na avaliação realizada pelo Algoritmo 34. Primeiramente, para cada recurso, o algoritmo seleciona sua primeira tarefa para ser escalonada, sendo que a tarefa é selecionada com base na prioridade calculada pelas heurísticas das Expressões 47 ou 49. Em seguida, dentre todas as próximas tarefas para serem escalonadas, o algoritmo escalona a tarefa com o tempo de conclusão mínimo (WANG et al., 2011). Ao final do algoritmo, o tempo de conclusão ftp_i é estimado para cada processador p_i do escalonamento, ao utilizar uma fila que_i que armazena e ordena todas as tarefas escalonadas para um processador p_i. No Algoritmo 34, a fila que_ready_i organiza todas as tarefas não escalonadas que estão prontas para executar no processador p_i (WANG et al., 2011).

As Equações 50 e 51, utilizadas pelo Algoritmo 34, correspondem ao tempo de início disponível e ao tempo de finalização de uma tarefa t_i. A Expressão 50 apresenta a estimativa do tempo de inicialização disponível, onde ftt_j é o tempo de finalização de uma tarefa t_j. Simultaneamente, a Expressão 51 estima o tempo de inicio fts_i e o tempo de finalização ftt_i^e de uma tarefa t_i, onde $ocioso(p_j)$ obtém o tempo de ociosidade de um processador p_j, $P(i)$ é o o processador onde a tarefa t_j foi designada e γ_j é velocidade de execução de instruções no processador p_j.

$$t_i^{avail} = \max_{e(j,i)\in E} ftt_j \tag{50}$$

Algoritmo 34 Algoritmo de Avaliação – adaptado de Wang et al. (2011)

entrada: cromossomo c;
saída: $\{fts_i, que_i\}$ para cada processador p_i;

para toda tarefa de entrada t_j **faça**
  adicione t_j para a fila de tarefas prontas $que\_ready_{c(j)}$:
  $t_j^{avail} \leftarrow 0$;
fim para
repita
  $min\_end \leftarrow \infty$; {Tempo de conclusão mínimo}
  $t\_sel \leftarrow$ vazio; {Tarefa selecionada}
  para todo processador p_i **faça**
    encontre em que_ready a tarefa t_j com prioridade máxima; {max-min fase 1}
    calcule ftt_j usando a Equação 51; {max-min fase 2}
    se $ftt_j < min\_end$ **então**
      $min\_end \leftarrow ftt_j$; {Atualizar o tempo de conclusão mínimo}
      $t\_sel \leftarrow j$; {Atualize a tarefa selecionada}
    fim se
  fim para
  $res\_sel \leftarrow c(t\_sel)$;
  remova t_{t_sel} de $que\_ready_{res\_sel}$;
  adicione t_{t_sel} para que_{res_sel};
  $fts_{res\_sel} = ocioso(res\_sel) = ftt_{t\_sel}$;
  para toda tarefa descendente t_i da tarefa t_{t_sel} **faça**
    calcule t_i^{avail} usando a Equação 50;
    se t_i está pronta pra executar **então**
      adicione t_i para $que\_ready_{c(i)}$;
    fim se
  fim para
até que_ready_i esteja vazia

$$\begin{aligned} fts_i &= max\{t_i^{\text{avail}}, ocioso(M(i))\} \\ ftt_i &= fts_i + |t_i|\gamma_j \end{aligned} \tag{51}$$

Após a obtenção do tempo de conclusão ftp_s para cada processador p_i, Wang et al. (2011) empregam a métrica da confiabilidade na tentativa de minimizar a quantidade de falhas, representada na Expressão 52, onde rdr_i é a taxa de falhas de um processador p_i. Nesta fase, o makespan pode ser obtido ao observar o tempo de conclusão máximo.

$$\sum_{i=1}^{m} ftp_i \cdot rdr_i \tag{52}$$

O terceiro GA apresentado por Hassan et al. (2015) utilizam uma técnica baseada na densidade local do grafo de tarefas para determinar a aptidão. Conforme os autores, seleciona-se o processador com tempo de conclusão mínimo, ftp_i. Após isso, dentre a lista tarefas disponíveis, considerando as restrições de precedência, escalona-se primeiro a tarefa com a maior quantidade de sucessores. Em seguida, a lista de tarefas disponíveis é

atualizada, de forma que uma nova busca pelo processador com menor ftp_i é reiniciada, repetindo o processo até o final da avaliação. Hassan et al. (2015) observam que escalonar uma tarefa que pode levar a novas tarefas para seu processador, pode afetar o FT de todo os sistema. Ao final, um escalonamento é formado, e FT pode ser obtido. As instruções utilizadas nessa avaliação são sumarizadas no Algoritmo 35, onde al é a lista de tarefas disponíveis sem predecessoras, sp é o processador como menor ftp_i selecionado e st é tarefa selecionada.

Algoritmo 35 Avaliação da Aptidão – adaptado de Hassan et al. (2015)

entrada: cromossomo c;
saída: tempo de finalização FT;
1: **enquanto** $|al| > 0$ **faça**
2: $sp =$ a máquina com o ftp_i mínimo;
3: $st =$ a tarefa com a maior quantidade de sucessores na sp;
4: Adicionar a st para a SP;
5: Atualizar ftp_i;
6: Atualizar al;
7: **fim enquanto**
8: **retorne:** FT $\{\max ftp_i\}$

O quarto GA, apresentado por Hassan et al. (2015), avalia os cromossomos semelhantemente ao Algoritmo 35, no entanto, emprega uma combinação com uma heurística de tempo mais ágil de inicialização. A avaliação da aptidão seleciona, dentre um conjunto AV de tarefas disponíveis sem tarefas predecessoras, a tarefa com ftp_j mínimo, caracterizando uma técnica de ECT. Em conjunto, a avaliação utiliza a densidade local do grafo ao também considerar a quantidade de tarefas sucessoras. Em resumo, a seleção de um tarefa $t_j \in AV$ é realizada conforme a Expressão 53, onde $\alpha \in [1, 0]$ e $|Succ_j|$ é a quantidade de sucessores de uma tarefa j (HASSAN et al., 2015).

$$\alpha \times (ftp_j) - (1 - \alpha) \times |Succ_j| \tag{53}$$

5.3 Função Objetivo na MLE

A função de aptidão empregada sobre a lista de mapeamentos engloba a ordenação topológica das tarefas e retorna o tempo de finalização do escalonador. O Algoritmo 36 demonstra a função de aptidão utilizada por Singh e Youssef (1996). Algumas funções essenciais utilizadas no algoritmo são:

- μ_i^{kmp}: indica o tempo de execução de uma tarefa i em uma máquina k usando o mapeamento mp, de maneira que $1 \leq k \leq m$ e $1 \leq mp \leq \phi$, onde ϕ representa a quantidade mapeamentos possíveis para as máquinas disponíveis;

- $\sigma(i_a^{mp}, j_b^{np}, l)$: indica o tempo de comunicação de uma tarefa i designada para uma máquina de tipo a usando um mapeamento mp para uma tarefa j designada à uma máquina de tipo b usando um mapeamento np sobre uma conexão l. Onde, para todo $i, j \in V$, $1 \leq a, b \leq r$, $1 \leq m, n \leq \phi$ e $1 \leq l \leq \lambda$.

Algoritmo 36 Cálculo de Aptidão – adaptado de Singh e Youssef (1996)

entrada: cromossomos c;
saída: tempo de finalização FT;
1: decodificar: $\forall i \in T, \pi(i) = (([I_i/\phi] + 1), (I_i \mod \phi + 1))$;
{Reordene os nós de forma que $i < j$, se i é predecessor de j}
2: Executa ordenamento topológico em G;
3: **para** $i = 1$; m **faça**
4: **se** i **não** possui predecessor **então**
5: $\sigma(i) = (0, \mu_i^{ab})$, onde $\pi(i) = (a, b)$;
6: **senão**
7: **para todo** todo processador p de i **faça**
8: $F_p = \min_{\forall links}[p_{fim} + \sigma(p_c^d, i_a^b, l)]$
$\{\sigma(p) = (p_{começo}, p_{fim})\}$
{Etapa computada nas interações anteriores devido a ordem topológica}
$\{\pi(p) = (c, d)\}$
9: $i_{inicio} = \max_{\forall p} F_p$
$\{i_{inicio}$ é o tempo de início do nó $i\}$
10: $i_{fim} = i_{inicio} + \mu_i^{ab}$;
11: $\sigma(i) = (i_{inicio}, i_{fim})$, onde $\pi(i) = (a, b)$;
12: **fim para**
13: **fim se**
14: **fim para**
15: **retorne:** tempo de finalização:$FT = \max_{\forall i \in T}(i_{fim})$

5.4 Função Objetivo na MSE

A etapa de avaliação realizada por Wang, Siegel e Roychowdhury (1996) e Wang et al. (1997) constrói o escalonamento considerando o custo de comunicação e a sequência de execução. O cenário adotado caracteriza cada máquina com um *link* de entrada e um *link* de saída de dados, sendo todas as máquinas interligadas por um *switch*. Inicialmente, o mecanismo escalona as tarefas de acordo com a cadeia de escalonamento, em que tarefas alocadas para um mesmo processador executam obedecendo suas respectivas posições na cadeia de escalonamento. De forma parecida, tarefas escalonadas para processadores diferentes tem sua execução controlada pela dependência de dados. Em adição, qualquer tarefa só pode ser escalonada quando todos os seus dados de entrada forem transferidos. Caso tarefas alocadas para um mesmo processador possuam dependência de dados, o custo de comunicação será zero. Após a montagem do escalonamento, o valor de aptidão

de um cromossomo é definido como o tempo de finalização da execução da última tarefa escalonada. Portanto, o valor de aptidão é o tempo de execução total, dado as cadeias de alocação e escalonamento.

O trabalho de Lee e Chen (2003) converte a otimização para maximização ao definir o valor de aptidão como $FT_{max} - FT$, onde FT define o tempo de execução total de um escalonamento e FT_{max} é o maior valor de makespan encontrado nos cromossomos da população.

Propondo dois GAs com duplicação de tarefas e baseados em prioridade, Yao, You e Li (2004) efetuam a computação da aptidão conforme sumarizado no Algoritmo 37. Dessa forma, tanto o GA padrão quanto o GA adaptado ao conceito de MS utilizam o mesmo procedimento avaliativo.

Algoritmo 37 Decodificação da MSE baseada em prioridade e com duplicação de tarefas – adaptado de Yao, You e Li (2004)

entrada: Cromossomo c;
saída: Comprimento do escalonamento FT;
1: Calcule o valor de pri de cada tarefa utilizando a Expressão 29;
2: Ordene a lista pri em ordem ascendente de valores pri;
3: **para toda** tarefa t na ordem ascendente de pri **faça**
4: **para todo** $p \in P = p_1, p_2, ..., p_m$ **faça**
5: **se** t não está alocado para p **então**
6: $ST(t,p) = 0$; $FT(t,p) = 0$;
7: **senão**
8: Considere todas as tarefas adiante de t na lista pri e custo de comunicação para determinar o mais rápido tempo de inicio de t, então $FT(t,p) = ST(t,p) + et(t)$;
9: **fim se**
10: **fim para**
11: **fim para**
12: **retorne:** $\max FT(t,p) | t \in T, p \in P$ como o comprimento de escalonamento;

Com as cadeias de escalonamento e alocação divididas em GAs distintos, Bonyadi e Moghaddam (2009) calculam um valor de aptidão para cada estrutura em cada tipo de população. Na população que armazena as cadeias de escalonamento, $popS$, uma cadeia de alocação é selecionada para cada cadeia de escalonamento, de forma que esse par selecionado minimize o makespan. O resultado dessa junção é atualizado como o valor de aptidão da cadeia de escalonamento, conforme apresentado no Algoritmo 38. De maneira análoga, conforme apresentado no Algoritmo 39, as cadeias da população de alocação, $popP$, também têm sua aptidão calculada na junção ideal com alguma cadeia de escalonamento, porém, executada no GA voltado às alocações.

Em seus dois primeiros algoritmos, Omara e Arafa (2009) efetuam a avaliação da aptidão ao combinar o tempo de finalização do escalonamento com o balanceamento de carga. Omara e Arafa (2009) justificam que várias soluções encontradas podem se igualar no makespan; entretanto, quando o balanceamento de carga é avaliado, nem todas as soluções

Algoritmo 38 Aptidão de cadeias de escalonamento – adaptado de Bonyadi e Moghaddam (2009)

entrada: *popS*, *pop*;
saída: Valores de aptidão para *popS*;

```
1: para toda sequência s em popS faça
2:    para toda cadeia de processamento p em popP faça
3:       Calcule o makespan do par (s, p) (chamado M_{s,p});
4:       se M_{s,p} é o melhor makespan encontrado para s então
5:          atualize a aptidão da sequência (s) como M_{s,p};
6:       fim se
7:    fim para
8: fim para
9: retorne: todos os valores de aptidão;
```

Algoritmo 39 Aptidão de cadeias de alocação – adaptado de Bonyadi e Moghaddam (2009)

entrada: *popS*, *popP*;
saída: Valores de aptidão para *popP*;

```
1: para toda sequência p em popP faça
2:    para toda cadeia s em popS faça
3:       Calcule o makespan do par (s, p) (chamado M_{s,p});
4:       se M_{s,p} é o melhor makespan encontrado para s então
5:          Atualize a aptidão da sequência (p) como M_{s,p};
6:       fim se
7:    fim para
8: fim para
9: retorne: todos os valores de aptidão;
```

apresentam bons resultados. Desse modo, a Expressão 54 calcula o tempo de finalização, onde a é uma constante e $comprimento(s)$ é o comprimento do escalonamento, sendo este último definido na Expressão 55. Paralelamente, o balanceamento de carga é calculado a partir da média do tempo de execução dos processadores, avg, e o tempo máximo de finalização, $comprimento(s)$, conforme apresentado na Expressão 56. O valor de avg é calculado conforme a Expressão 57, onde o tempo de execução de cada processador p_j é obtido por $ftp[p_j]$.

$$f = (\frac{a}{comprimento(s)}) \tag{54}$$

$$comprimento(s) = \max(ftt(t,)), i = 1, ..., T \tag{55}$$

$$LoadBalance = \frac{comprimento(s)}{avg} \tag{56}$$

$$avg = \sum_{j=1}^{m} \frac{ftp[p_j]}{m} \tag{57}$$

O trabalho de Chitra et al. (2010) é um outro exemplo de otimização com mais de um objetivo, de forma que três métricas são consideradas: makespan, flowtime e confiabilidade. Inicialmente, o makespan pode ser obtido ao encontrar o maior tempo de conclusão dentre os processadores disponíveis, conforme apresentado na Expressão 58, e o flowtime caracteriza a soma do tempo de finalização de todas as tarefas. Chitra et al. (2010) observam que o flowtime possui uma magnitude maior que o makespan e sua diferença aumenta ao acréscimo de tarefas e máquinas. Desse modo, a Expressão 59 demonstra o cálculo do flowtime médio, $aft(s)$.

$$FT = \max_j ftp_j(s) \tag{58}$$

$$aft(s) = \frac{\sum_j ftp_j(s)}{P} \tag{59}$$

A terceira métrica otimizada por Chitra et al. (2010) é a confiabilidade, sendo essa observada em um cenário em que os processadores tratam as falhas de execução transitoriamente, isto é, uma falha em um processador só afeta a tarefa em execução no momento e não suas tarefas descendentes. A Expressão 60 obtém a probabilidade de um processador p_j executar todas as suas tarefa sucessivamente, onde λ é uma constante, seguindo o modelo de distribuição de Poisson. Em adição, considerando as falhas como fenômenos independentes, a Expressão 61 calcula a probabilidade de um escalonamento s finalizar corretamente. Por fim, o índice de confiabilidade pode ser definido conforme a Expressão 62.

Em um estudo comparativo, Chitra, Venkatesh e Rajaram (2011) utilizam as expressões 58 e 62 para avaliar a aptidão dos indivíduos, representando respectivamente o makespan e a confiabilidade.

$$p^j_{succ}(s) = e^{-\lambda_j ftp_j(s)} \tag{60}$$

$$p_{succ} = e^{-\sum_j \lambda_j ftp_j(s)} \tag{61}$$

$$rel(s) = \sum_j \lambda_j ftp_j(s) \tag{62}$$

Para avaliar os objetivos de makespan e confiabilidade, Sathappan et al. (2011) utilizam uma abordagem baseada em soma ponderada. Desse modo, o makespan é obtido pela Expressão 58 e o índice de confiabilidade, pela Expressão 62. Ambos são ponderados de acordo com sua importância. A Expressão 63 apresenta a combinação das duas funções, onde ω é um valor, entre 0 e 1, que atua como coeficiente ponderativo. O valor do mínimo e máximo são representado por FT_{min} e FT_{max}, respectivamente. Da mesma forma, o valor do menor e do maior índice de confiabilidade é dado por CF_{min} e CF_{max}. Sathappan

et al. (2011) destacam que, ao variar os valores possíveis de ω, o balanceamento entre o makespan é a confiabilidade pode ser determinado. Além disso, ao utilizar combinações lineares de ambos os objetivos, o problema bi-objetivo pode ser convertido para um único objetivo.

$$f(s) = \omega \cdot \frac{FT - FT_{min}}{FT_{max} - FT_{min}} + (1 - \omega) \cdot \frac{CF - CF_{min}}{CF_{max} - CF_{min}} \tag{63}$$

A Expressão 64 é utilizada como forma de avaliar a aptidão no trabalho de Gupta, Kumar e Agarwal (2010), de forma que $FT(i)$ é o tempo de finalização do i-ésimo cromossomo; FT_{max} e FT_{min} correspondem ao tempo de finalização máximo e mínimo da atual população de cromossomos, respectivamente.

$$F(i) = (FT_{max} - FT(i) + 1/(FT_{max} - FT_{min} + 1)) \tag{64}$$

No trabalho de Kang, Zhang e Chen (2011), a métrica do makespan é empregada para a avaliação de aptidão, sendo obtida após a leitura do escalonamento gerado pela decodificação do cromossomo.

Mohamed e Awadalla (2011), Awadall, Ahmad e Al-Busaidi (2013) traduzem a otimização do problema para minimização do makespan através da Expressão 65, onde FT_{max} é o tempo de finalização máximo encontrado na população, e $ftt(t)$ é o tempo de finalização de uma tarefa t.

$$f = FT_{max} - \max_{t \in T} ftt(t) \tag{65}$$

Os dois GAs apresentados por Ahmad, Munir e Nisar (2012) possuem métodos diferentes para avaliar a aptidão. O primeiro GA, define a aptidão como $f = c/makespan$, onde c é uma constante. Já o segundo GA utiliza o Algoritmo 40 para decodificar os cromossomos e obter o valor de aptidão.

Algoritmo 40 Determinação do comprimento do escalonamento – adaptado de Ahmad, Munir e Nisar (2012)

entrada: Cadeia de alocação do cromossomo;
saída: Comprimento do escalonamento FT;
1: **para toda** t_i na cadeia de alocação **faça**
2: Identifique o processador p_j da cadeia de alocação alocado para t_i;
3: Determine o tempo de finalização de t_i em p_j;
4: Elimine t_i da cadeia de escalonamento;
5: **fim para**
6: FT = tempo de finalização máximo;
7: **retorne:** FT

De forma análoga ao primeiro GA apresentado por Ahmad, Munir e Nisar (2012), os trabalhos de Kaur e Singh (2012), Panwar, Lal e Singh (2012) e Ahmad et al. (2016) utilizam a divisão $1/comprimento(s)$ para calcular a aptidão.

A Expressão 66 apresenta a ponderação de dois objetivos realizada por Dhingra, Gupta e Biswas (2014), onde f_1 representa o tempo de máximo de conclusão, makespan, f_2 representa a soma dos tempos de conclusão de cada tarefas, flowtime, e α é um coeficiente de ponderação que varia de 0 a 1.

$$f = \min(\alpha f_1 + (1 - \alpha) f_2) \tag{66}$$

Adicionalmente à normalização supracitada, ao final de cada geração, Ahmad et al. (2016) analisam soluções em busca de problemas quanto ao balanceamento de carga. Desta forma, 50% dos cromossomos têm a qualidade de seus escalonamentos elevada ao balancear de carga dos processadores do conjunto. A Expressão 67 é utilizada para determinar o índice de balanceamento pra cada solução, onde $comprimento(s)$ é o comprimento do escalonamento e ftp_j é o tempo de finalização de um processador j. Ahmad et al. (2016) alertam que, quanto maior for o valor de LB, pior é o balanceamento de carga.

$$LB = comprimento(s) - \min\{ftp_1, ftp_2, ftp_3, ..., ftp_m\} \tag{67}$$

5.5 Função Objetivo na OLE

A primeira abordagem utilizada para a definição da aptidão de um cromossomo na representação OLE é apresentada no trabalho de Kwok e Ahmad (1997). A Expressão 68 apresenta a equação para obtenção da aptidão, onde $comprimento(s)$ determina o comprimento do escalonamento. Ao final do cálculo, o valor da aptidão estará padronizado entre 0 e 1.

$$\frac{(\sum_i^n et(t_i) - comprimento(s))}{\sum_i^n et(t_i)} \tag{68}$$

No trabalho de Ramachandra e Elmaghraby (2006), a aptidão é computada como $\sum et(t_j) ftt_j$, onde $et(t_j)$ representa o custo da tarefa e ftt_j o tempo de finalização de uma tarefa j. Para definição do tempo de finalização, Ramachandra e Elmaghraby (2006) calculam escalonamento através da regra FAM, em que tarefas são escalonadas observando as primeiras máquinas disponíveis.

A função de aptidão é implementada no trabalho de Xu et al. (2012) juntamente ao algoritmo de alocação HEFT (TOPCUOGLU; SEVILMIS, 2002). Nesse caso, o makespan é definido com o tempo de finalização da última tarefa processada. O Algoritmo 41 sumariza as instruções utilizadas no trabalho de Xu et al. (2012) para o cálculo da aptidão, de forma que $EFT(t_i, p_k)$ define o mais rápido tempo de finalização de uma tarefa t_i no processador p_k. Além de implementar o Algoritmo 41 para alocar as tarefas, o trabalho

de Xu et al. (2014) adapta a otimização em forma de minimização conforme a Expressão 69.

$$f_i = \max_{j \in pop}(FT_j) - FT_i + 1 \tag{69}$$

Algoritmo 41 Cálculo da Aptidão baseada em HEFT – adaptado de Xu et al. (2012) e Xu et al. (2014).

entrada: Fila de escalonamento (cromossomo);
saída: FT;
1: Preencha a Fila de escalonamento com as tarefas;
2: **enquanto** Fila de escalonamento $\neq \emptyset$ **faça**
3: Selecione a primeira tarefa t_i da Fila para escalonamento;
4: **para todo** processador p_k no conjunto de processadores **faça**
5: Calcule $EFT(t_i, p_k)$ usando a política de inserção baseada em HEFT;
6: Aloque a tarefa t_i par ao processador p_k que minimiza $EFT(t_i)$;
7: **fim para**
8: Remove t_i da Fila de Escalonamento
9: **fim enquanto**
10: **retorne:** aptidão = $FT = EFT(T_{exit})$;

Conforme supracitado, o primeiro GA considerado no trabalho de Hassan et al. (2015) trabalha sob a codificação OLE. Neste contexto, a função objetivo é obtida utilizando a técnica de menor tempo de conclusão (ECT[1]), segundo a qual a tarefa é direcionada ao processador em que tal tarefa possua o tempo de conclusão mínimo.

5.6 Função Objetivo na TE

A função objetivo no trabalho de Topcuoglu e Sevilmis (2002) é calculada conforme a Expressão 70, onde $NCost$ identifica custo normalizado de um indivíduo e $NCost_{max}$ é o custo normalizado máximo encontrado na população.

$$f = NCost_{max} - NCost + 0.01 \times NCost_{max} \tag{70}$$

Ao empregar a codificação baseada em TE com diminuição de restrições, o trabalho de Wu et al. (2004) aborda a utilização de cromossomos com soluções inválidas. Desta forma, a função objetivo deve ser adaptada para penalizar soluções impossíveis. A Expressão 71 apresenta o soma ponderada empregada por Wu et al. (2004) na avaliação das soluções, onde $0.0 \leq b \leq 1.0$. A variável $task\_fitness$ recompensa soluções válidas, enquanto que, se uma solução for válida, a variável $processor\_fitness$ recompensará soluções mais curtas.

$$f = (1 - b) \times task\_fitness + b \times processor\_fitness \tag{71}$$

[1] Do inglês, *Earliest Completion Time*.

Wu et al. (2004) alertam que soluções inválidas não têm o valor de $processor\_fitness$ calculado, o que penaliza a solução, conforme a Expressão 72.

$$fitness = (1 - b) \times task\_fitness + b \times 0 = (1 - b) \times task\_fitness \tag{72}$$

O cálculo utilizado no componente $task\_fitness$ utiliza algumas orientações. Para que um escalonamento de um par de tarefas em um único processador seja válido, as tarefas do par devem ser independentes ou a ordem de alocação do processador deve corresponder a ordem de dependência (WU et al., 2004). Portanto, uma solução só pode ser totalmente válida se todos os processadores possuírem escalonamentos válidos. Além disso, Wu et al. (2004) elaboraram o $task\_fitness$ de forma que seja uma avaliação gradual, isto é, pequenas sequências com ordenações válidas são recompensadas, e eventualmente, ao incrementar o comprimento das sequências, soluções inteiramente válidas poderão ser encontradas. Ao empregar esses conceitos, Wu et al. (2004) dividem o componente $task\_fitness$ em dois componentes principais: a porcentagem do total de tarefas codificadas por um indivíduo e a porcentagem de sequências válidas dado um comprimento máximo. A Expressão determina a relação entre os componentes de $task\_fitness$:

$$task\_fitness = raw\_fitness \times task\_ratio \tag{73}$$

Inicialmente, para o cálculo do $raw\_fitness$, é necessário definir o componente que controla a evolução gradual. Nesse caso, Wu et al. (2004) definem a evolução como eras, na forma: $era = 0, 1, 2...., E$. O valor máximo definido para a era, $E \leq T$, é definido pelo usuário e, em um primeiro momento, $era = 0$. A evolução gradual é executa ao incrementar o valor de era em duas situações: quando a média de aptidão da população excede um limite definido pelo usuário, $thresh$, e quando o número de cromossomos com a aptidão máxima excede outro limite definido pelo usuário, $thresh\_maxfit$ (WU et al., 2004).

Para o cálculo de $raw\_fitness$, dois intervalos são considerados: o primeiro, identificado como as primeiras $era + 1$ (ou anteriores) tarefas alocadas, e o segundo, onde todas as sequências de comprimento $era + 2$. A importância do primeiro intervalo da-se ao incremento de era, de forma que a probabilidade de um processador conter menos que $era + 2$ tarefas aumenta (WU et al., 2004).

Inicialmente, a Expressão 74 é utilizada para determinar a pontuação das tarefas, no intervalo $era + 1$ (ou anterior) em um único processador. A função $numtasks(p)$, onde

$p = 1, ..., P$, indica o número de tarefas alocadas a um processador p.

$$subseq(p) = \begin{cases} 1 & \text{se } numtasks(p) > 0 \\ 0 & \text{caso contrário} \end{cases}$$
$$valseq(p) = \begin{cases} 1 & \text{se as tarefas da primeira } era + 1 \text{ (ou menos) do processador } p \text{ estão em ordem válida} \\ 0 & \text{caso contrário} \end{cases} \quad (74)$$

Conforme a definição de Wu et al. (2004), as equações na Expressões 74 referem-se a processadores individuais. A Expressão 75 resume o cálculo para obter os valores, referentes às sequências, para todos os processadores de um cromossomo.

$$Subseq = \sum_{p=1}^{P} subseq(p), \quad Valseq = \sum_{p=1}^{P} subseq(p) \quad (75)$$

Em seguida, para determinar a pontuação referente as sequências, no intervalo $era+2$, para um único processador, utiliza-se as equações presentes na Expressão 76.

$$\begin{aligned} se(p) &= \#\text{ de sequências de comprimento } era + 2 \text{ no Processador } p \\ ve(p) &= \#\text{ de sequências válidas de comprimento } era + 2 \text{ no Processador } p \end{aligned} \quad (76)$$

De maneira análoga ao espaço $era + 1$, ao combinar as equações da Expressão 76, é possível computar a pontuação, para cada cromossomo, referente as sequências dispostas no intervalo $era + 2$, conforme demonstrado na Expressão 77.

$$SE = \sum_{p=1}^{P} se(p), \quad VE = \sum_{p=1}^{P} ve(p) \quad (77)$$

Ao final, o $raw\_fitness$ de um cromossomo é obtido conforme a Expressão 78.

$$raw\_fitness = \frac{Valseq + VE}{Subseq + SE} \quad (78)$$

Em contrapartida, a contagem de tarefas codificadas pelo cromossomo é definida por $task\_ratio$. O cálculo obtém a porcentagem de tarefas distintas mediante a todas as tarefas especificadas no problema (Expressão 79). Sendo assim, $task\_ratio$ penaliza cromossomos que não contenham todas as soluções, o que pode encorajar o sistema a incluir uma cópia de cada indivíduo no cromossomo (WU et al., 2004).

$$task\_ratio = \frac{\text{quantidade de tarefas distintas codificadas no cromossomo}}{\text{total de tarefas no problema}} \quad (79)$$

O valor de $processor\_fitness$ só pode ser calculado caso a solução seja viável, caso contrário, $processor\_fitness$ recebe zero. Ao calcular o valor de $processor\_fitness$, ts

representa o tempo de execução encontrado pelo escalonamento codificado. Em seguida, o tempo necessário para executar todas as tarefas em um único processador é definido por $serial\_len$, enquanto que $super\_serial\_len = P \times serial\_len$, onde P é igual a quantidade de processadores. Conforme observado por Wu et al. (2004), qualquer solução viável deve ser $ts \ll super\_serial\_len$, o que torna $super\_serial\_len$ um seguro limite superior para o tempo de execução. Desta forma, o objetivo é minimizar ts, conforme demonstrado na Expressão 80.

$$processor\_fitness = \frac{super\_serial\_len - ts}{super\_serial\_len} \tag{80}$$

Apesar que, teoricamente, o valor máximo para $processor\_fitness$ é 1, esse valor não pode ser encontrado, uma vez que o tempo de execução, ts, deveria ser igual a zero, e portanto, as tarefas precisam de mais do que zero unidades de tempo para serem completadas.

O trabalho de Demiroz e Topcuoglu (2006) combina a minimização de dois objetivos: comprimento de escalonamento (makespan) e quantidade de processadores utilizada. Essa combinação voltada a minimizar o custo normalizado, $NCost$, é demonstrada na Expressão 81, onde NSL[2] é o comprimento do escalonamento normalizado, e NPU[3] é a utilização normalizada dos processadores. Os itens w_1 e w_2 são coeficientes associados a importância dos objetivos.

$$NCost = w_1 \times NSL + w_2 \times NPU \tag{81}$$

Demiroz e Topcuoglu (2006) destacam que a normalização do comprimento do escalonamento é realizada pois um grande conjunto de grafos, com diferentes propriedades, foi considerado nos experimentos. A Expressão 82 demonstra como o NSL pode ser calculado, onde CP_{min} representa o caminho crítico de um grafo caso o custo de computação de cada tarefa seja atribuído ao valor mínimo nas várias alternativas de processador (DEMIROZ; TOPCUOGLU, 2006). O comprimento do escalonamento, denominado na Expressão 82, representa a soma dos custos de computação mínimos de cada tarefa, onde $peti, j$ representa o custo computacional de executar a tarefa t_i em um processador p_j.

$$NSL = \frac{\text{Comprimento Escalonamento}}{\sum_{t_i \in CP_{min}} \min_{p_j \in P} pet\{i, j\}} \tag{82}$$

Ainda sobre a otimização proposta por Demiroz e Topcuoglu (2006), o elemento NSL pode ser computado conforme a Expressão 83, onde número total de unidades de processamento é representado por P. O comprimento de um grafo caracteriza o máximo grau de concorrência, que é a quantidade máxima de tarefas que podem ser executadas paralelamente em um gráfico (DEMIROZ; TOPCUOGLU, 2006). No grafo da Figura 1,

[2] Do inglês, *Normalized Schedule Length*.
[3] Do inglês, *Normalized Processor Usage*.

dentre outras possibilidades de paralelismo, as tarefas t_3, t_4 e t_5 configuram ao grafo um comprimento igual a 3.

$$NPU = \frac{P \text{ usados}}{\text{Comprimento do grafo}} \tag{83}$$

A avaliação de aptidão empregada no terceiro algoritmo de Omara e Arafa (2009) lida com cromossomos válidos e inválidos. Dessa forma, o critério de punição adotado para cromossomos inválidos é atribuir o valor zero como nota final. Os demais cromossomos válidos recebem como aptidão um valor normalizado definido na Expressão 84, onde $comprimento(s)$ define o tempo de finalização máximo encontrado dentre as tarefas do grafo, isto é, o makespan.

$$f = 1/comprimento(s) \tag{84}$$

Com o objetivo de obter um bom balanceamento de carga adotando a codificação baseada em TE, Pop, Dobre e Cristea (2009) dividem a função objetivo em quatro fatores de medição: f_1, f_2, f_3 e f_4. O primeiro fator avaliado, para um cromossomo c, f_1, é demonstrado na Expressão 85, onde $|c|$ representa o número de tarefas do cromossomo c. Dessa forma, o primeiro fator considerado verifica se o escalonamento está perfeitamente balanceado, isto é, convergindo para 1 quando t_m se aproxima de t_M.

$$f_1 = \frac{t_m}{t_M} = \frac{min_{1 \leq i \leq |c|}\{t_i\}}{max_{1 \leq j \leq |c|}\{t_j\}}, 0 \leq f_1 \leq 1 \tag{85}$$

Posteriormente, o segundo fator avaliado, f_2, verifica a média de utilização dos processadores, sendo demonstrado na Expressão 86. No caso ideal, os totais de execução nos processadores são iguais entre si e iguais ao makespan, o que leva ao valor 1 na média de utilização dos processadores (POP; DOBRE; CRISTEA, 2009).

$$f_2 = \frac{1}{c} \sum_{j=1}^{|c|} \frac{t_j}{t_M}, 0 \leq f_2 \leq 1 \tag{86}$$

Em seguida, o terceiro fator observado pela avaliação de Pop, Dobre e Cristea (2009), f_3, verifica se as tarefas do escalonamento atendem as restrições impostas (ordem, prazo, recursos). A Expressão 87 apresenta o cálculo desse fator, onde $T_s(c)$ denota, para um cromossomo c, as tarefas do grafo que atendem as restrições, e T é atribuído a quantidade total de tarefas.

$$f_3 = \frac{T_s(c)}{T}, 0 \leq f_3 \leq 1 \tag{87}$$

Por fim, o quarto e último fator, f_4, considerado por Pop, Dobre e Cristea (2009), avalia as dependências entre as tarefas, sendo a avaliação inteiramente relacionada ao comprimento do escalonamento. O cálculo do quarto fator é apresentado na Expressão 88, onde $SL(c)$ determina o comprimento do escalonamento codificado pelo cromossomo

c, e t_M denota o comprimento máximo de escalonamento (makespan) para todos os cromossomos da população.

$$f_4 = \frac{SL(c)}{t_M} \tag{88}$$

Em resumo, Pop, Dobre e Cristea (2009) combinam os fatores supracitados em uma única expressão para calcular a aptidão de cada cromossomo. Essa combinação dos fatores é apresentada na Expressão 89.

$$\begin{gathered} F(c) = f_1 \times f_2 \times f_3 \times f_4 \\ F(c) = (\frac{t_m}{t_M}) \times (\frac{1}{c}\sum_{j=1}^{|c|}\frac{t_j}{t_M}) \times (\frac{Ts(|c|)}{T}) \times (\frac{SL(c)}{t_M}), 0 \leq F(c) \leq 1 \end{gathered} \tag{89}$$

A função objetivo empregada no trabalho de Amalarethinam e Selvi (2012) combina três parâmetros para otimização, sendo o cálculo apresentado na Expressão 90. Os parâmetros são: FT, o makespan, P, a quantidade de recursos (processadores), e ARI[4], a média de tempo ocioso dos recursos.

$$f = \frac{1}{FT} + \frac{P}{ARI} \tag{90}$$

A otimização empregada pelo trabalho de Akbari, Rashidi e Alizadeh (2017) e Akbari (2018) combina dois objetivos: a minimização do makespan e a maximização da paralelização. Desta forma, a função multiobjetivo (MOF[5]) utilizada é apresentada na Expressão 91, onde λ é um valor aleatório, e β determina o valor de paralelização.

$$MOF = \lambda \times \text{makespan} + (1 - \lambda) \times \frac{1}{\beta}, 0 \leq \lambda \leq 1 \tag{91}$$

Conforme a definição de Akbari, Rashidi e Alizadeh (2017), com $\lambda = 0$, o escalonamento considera somente a maximização da paralelização. Caso contrário, com $\lambda = 1$, a minimização do makespan é enfatizada. O cálculo de β é apresentado na Expressão 92, de forma que $|p_i|$ representa a quantidade de tarefas alocadas ao processados p_i.

$$\beta = \frac{1}{\sum_{i=0}^{p-1} |\frac{n}{m} - |p_i||} \tag{92}$$

Conforme a observação de Amirjanov e Sobolev (2017), uma escalonamento é finalizado quando não restam mais tarefas ou quando este encontra um impasse. Portanto, a aptidão, implementada no trabalho de Amirjanov e Sobolev (2017), é estruturada em um formato que permite avaliação mediante punição, uma vez que tanto cromossomos válidos ou inválidos podem ser avaliados. O cálculo para aptidão é apresentado na Expressão 93,

[4] Do inglês, *Average Resource Idle Time*.
[5] Do inglês, *Multi-objective function*.

onde ϕ é a aptidão obtida, f é função objetivo, e k é um coeficiente para a função de penalidade ψ.

$$\phi = f + k\psi \tag{93}$$

Dentre os parâmetros da aptidão utilizada por Amirjanov e Sobolev (2017), a função objetivo f, que representa o makespan do escalonamento, é demonstrada na Expressão 94, onde n é quantidade de tarefas do grafo e n_{sh} é a quantidade de tarefas que foram escalonadas até que o processo de escalonamento foi finalizado, portanto, quando um impasse for encontrado ou quando todas as tarefas foram escalonadas (AMIRJANOV; SOBOLEV, 2017).

$$f = f_{sh} + \frac{f_{sh}}{n_{sh}}(n - n_{sh}) = f_{sh}\frac{n}{n_{sh}} \tag{94}$$

O componente f_{sh} utilizado na função objetivo representa o tempo máximo de finalização dentre todas as tarefas, sendo expressado na Equação 95, onde $T_{sh} \subseteq T$ e contempla todas as tarefas escalonadas.

$$f_{sh} = \max_{t \in T_{sh}}\{stt(t) + et(t)\} \tag{95}$$

Por fim, a penalidade imposta no cálculo da aptidão pune qualquer cromossomo que contenha tarefas não escalonadas. A função de penalidade ψ é apresentada na Expressão 96, onde w_{av} é a média do tempo de execução das tarefas no grafo. Portanto, se todas as tarefas foram escalonadas, naturalmente, o valor da função de penalização ψ será igual a 0 (AMIRJANOV; SOBOLEV, 2017).

$$\psi = w_{av}(n - n_{sh}) \tag{96}$$

5.7 Função Objetivo na DLE, FTE e PE

Em relação à DLE, o trabalho de Liu, Li e Yu (2002) adotam a métrica do tempo de finalização do escalonamento, incluindo o tempo de execução e o tempo de atraso da rede. Dado um escalonamento s, o valor de aptidão é definido como: $f(s) = FT_c - FT_{max}$, onde FT_c é tempo de execução serial de todas as tarefas em um processador local e FT_{max} é o tempo de finalização máximo encontrado dentre as tarefas do grafo (LIU; LI; YU, 2002).

Já a métrica empregada pela codificação FTE para medir o desempenho das soluções é o makespan, onde o tempo total de execução é formado pelas restrições de precedência em conjunto ao custo de comunicação entre os processadores. Segundo o trabalho de Jelodar et al. (2006), ao considerar as diversas cópias de cada tarefa precedente, o tempo mínimo de execução de cada tarefa precedente é calculado. Essa operação é realizada para que se possa definir o tempo de inicialização da tarefa descendente.

Finalmente, na codificação por lista de prioridades (PE), o valor da função da aptidão é computado após o cálculo do Gráfico de Gantt (HWANG; GEN; KATAYAMA, 2008). Desta forma, a função de avaliação emprega a métrica do makespan do escalonamento. Além disso, uma vez que o makespan reflete um objetivo de minimização, uma conversão para maximização pode ser realizada, conforme a Expressão 97. Simultaneamente, o trabalho de Azghadi et al. (2008) considera a minimização do tempo total de execução como otimização.

$$f(c_k) = 1/FT^k \qquad k = 1, ..., pop \tag{97}$$

5.8 Função Objetivo na DVFSE

O trabalho de Guzek et al. (2014) atua sobre duas métricas: makespan e consumo elétrico. Para otimizar o makespan, as tarefas são escalonadas utilizando uma heurística de inserção. O que define a ordem de prioridade das tarefas são os índices determinados por *b-level*, onde cada índice identifica o caminho mais longo desde o começo da tarefa até o final do grafo. Posteriormente, uma coleção ordenada das tarefas é formada, de forma que tarefas com maiores valores de *b-level* são escalonadas primeiro. Ao final, a heurística seleciona tarefas dispostas no inicio da coleção, verifica quando é possível escalonar a tarefa para o processador designado e escalona a tarefa para a primeira posição possível (GUZEK et al., 2014).

O segundo objetivo empregado em Guzek et al. (2014) é relacionado ao consumo elétrico, sendo baseado na equação de consumo elétrico dinâmico de um circuito CMOS, conforme a Expressão 98, onde A é o número de trocas por ciclo de relógio, C_{ef} é a capacidade total de carga, V é voltagem fornecida, e fe é a frequência correspondente.

$$P = AC_{ef}V^2f = \alpha V^2 fe \tag{98}$$

Uma vez que A e C_{ef} são constantes em uma máquina, Guzek et al. (2014) simplificam A e C_{ef} para um coeficiente alfa atribuído como 1 para normalização das tabelas de frequência e voltagem. Outra modificação é demonstrada ao substituir a frequência fe pela velocidade relativa rs_j, em consequência de rs_j ser proporcional à fe, e por rs_j ser apresentado pelos desenvolvedores de processadores.

Por fim, o consumo elétrico é calculado conforme a Equação 99, onde x é o tempo, $P_l(x)$ é a carga de uma máquina r_l sobre o tempo, $v_l(x)$ é a voltagem da máquina r_l sobre o tempo, $s_l(x)$ é o velocidade relativa da máquina r_l sobre o tempo e dx representa o par DVFS (GUZEK et al., 2014). Adicionalmente, é empregada uma técnica de recuperação de folga para reduzir o consumo elétrico sem aumentar o makespan. Essa técnica consiste em utilizar o intervalo, espaço entre o final de uma tarefa e o início da próxima tarefa

para combinar o par DVFS com o menor par possível, consequentemente, reduzindo o consumo de energia total.

$$E_t = \sum_{j=0}^{m} \int_0^{FT} P_l(x)dx = \sum_{l=0}^{m} \int_0^{FT} v_l(x) S_l(x)^2 dx \tag{99}$$

Os trabalhos de Sheikh, Ahmad e Fan (2016), Sheikh, Ahmad e Arshad (2017) propõem a otimização de três objetivos: makespan, consumo elétrico e temperatura do sistema, apresentadas respectivamente nas Expressões 100, 101 e 102. O makespan é obtido observando-se o tempo de finalização FT (de forma que ftt_i é o tempo de finalização da tarefa), ao passo que o consumo de energia é obtido sobre a energia gasta enquanto se executa uma tarefa combinada com tempo necessário para sua execução (respectivamente, en_i e et_i). O último objetivo, temperatura, é construído sobre a tentativa de minimizar, dentre todas as tarefas e processadores, a temperatura máxima $Temp_i^j$ obtida no j-ésimo núcleo de processamento consequente a alocação da i-ésima tarefa.

$$\text{Minimizar } FT = \max_{1 \leq i \leq n} ftt_i \tag{100}$$

$$\text{Minimizar } \sum_{i=1}^{n} en_i et_i \tag{101}$$

$$\text{Minimizar } \max_{1 \leq i \leq n} \max_{1 \leq j \leq m} Temp_i^j \tag{102}$$

Adicionalmente, as Expressões 103, 104 e 109 mostram o cálculo dos componentes utilizados em Sheikh, Ahmad e Fan (2016). Para calcular o tempo requirido para execução de uma tarefa, et_i, a Expressão 103 utiliza fe_0, a mais alta frequência disponível, e et_i^0, o tempo de execução da i-ésima tarefa em fe_0 (o peso da tarefa no grafo). O valor de fe_i, o nível de frequência correspondente, pode ser obtido por $fe_i = \gamma(v_i)^{\gamma^0}$, onde v_i é o nível de voltagem empregado, γ e γ_0 são constantes referentes ao nível de voltagem e frequência de um núcleo.

$$et_i = (fe_0/fe_i)et_i^0 \tag{103}$$

Já a Expressão 104 calcula o consumo elétrico de um núcleo ao processar uma tarefa t_i, onde p_{static} representa a energia dissipada pelo núcleo quando ocioso devido a corrente de fuga, γ' é uma constante dependente da tecnologia, onde seu valor geralmente é entre $2-3$ e C_{eff} representa a capacitância efetiva.

$$p_i = p_{static} + C_{eff}(v_i)^{\gamma'} \tag{104}$$

Adicionalmente, considerando PD_i como o conjunto de tarefas predecessoras de uma tarefa t_i, o tempo de início de uma tarefa t_i pode ser calculado conforme a Expressão 105, onde $d(pred_j, i)$ é o tempo requerido para transferir dados entre $pred_j$ e a tarefa t_i. Já

a Expressão 106 calcula o tempo de conclusão de uma tarefa t_i ao considerar seu menor tempo de inicialização.

$$\begin{gathered} \forall pred_j \in PD_i \\ st_i = max_{pred_j}[ft_{pred_j} + d(pred_j, i)] \end{gathered} \tag{105}$$

$$ft_i = st_i + et_i \tag{106}$$

Finalmente, para calcular a temperatura dos núcleos, Sheikh, Ahmad e Fan (2016) estimam o tempo de sobreposição das tarefas e o efeito vizinho. Para cada par de tarefas, uma variável determina seu tempo de sobreposição enquanto executam nos núcleos, conforme a Expressão 107, onde fts_i indica o tempo de inicialização de uma tarefa t_i.

$$\begin{gathered} \forall 1 \leq i, j \leq n \wedge i \neq j \\ z_{i,j} = \begin{cases} 1 & fts_j \leq fts_i \leq ftt_j \\ 0 & \text{caso contrário} \end{cases} \end{gathered} \tag{107}$$

O efeito vizinho é o impacto de dissipação de energia dos núcleos vizinhos na temperatura de um núcleo em particular. Primeiramente, sem considerar o efeito vizinho, calcula-se a temperatura de todos os núcleos, conforme a Expressão 108, onde R_{th} é a resistência térmica, T_A é a temperatura ambiente e $x_{i,j}$ indica se a i-ésima tarefa executa no j-ésimo núcleo.

$$\begin{gathered} \forall 1 \leq i \leq n, 1 \leq j \leq m \\ \alpha_i^j = R_{th} p_i x_{i,j} + T_A \end{gathered} \tag{108}$$

Ao final, ao utilizar a variável $z_{i,j}$ para implementar e acrescentar o efeito vizinho, a temperatura de cada núcleo pode ser obtida conforme a Expressão 109, onde as decisões de alocações da i-ésima e r-ésima tarefas são representadas pelos vetores x_i e x_r. $\mathbf{B}$ é matriz simétrica que contém todos os valores de $\beta_{(a,b)}$, relação de energia de um núcleo (a) quanto a temperatura de um núcleo vizinho (b), onde $\beta_{(a,b)} = 0 \; \forall a = b$, e $1 \leq a, b \leq m$.

$$T_i^j = \alpha_i^j + \sum_{r=1}^{n} Z_{i,r}(x_i \mathbf{B} x_r^T) p_r \tag{109}$$

5.9 Função Objetivo na RE, ME e QE

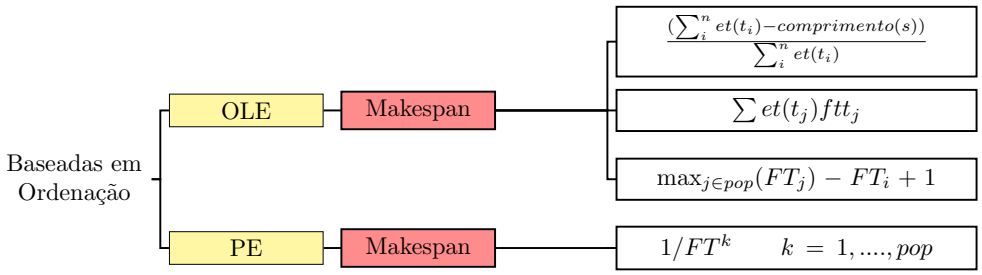
Os estudos que contemplam as codificações de Intervalo (RE), matrizes (ME) e Q-bit (QE) (WANG; KORFHAGE, 1995; HASSAN et al., 2015; GANDHI; NITIN; ALAM, 2017) não apresentam explicitamente as funções objetivo empregadas. Entretanto, a otimização em todos os casos é baseada na minimização do makespan.

5.10 Sumário

Esta seção sintetiza as funções objetivo apresentadas no decorrer do capítulo. Ao contrário da ordem cronológica adotada nas seções, essa síntese distribui as funções objetivo conforme a organização do Capítulo 4, ou seja, as representações identificadas e suas respectivas funções objetivo estão reunidas nas categorias de codificações baseadas em ordenação, alocação, alocação combinada com ordenação e outras codificações. Além disso, esse resumo agrupa as funções objetivo apresentadas explicitamente pelos autores dos trabalhos, isto é, funções objetivo descritas somente por textos ou baseadas exclusivamente sobre algoritmos não são listadas. Contudo, essas funções são encontradas nas suas respectivas seções neste capítulo.

Inicialmente, a Figura 89 apresenta as funções objetivo levantadas nas codificações baseadas em ordenação, neste caso, as representações OLE e PE. Em adição, a figura também distribui as funções considerando a métrica avaliada. Em tal cenário, em ambas as representações, a métrica prevalente é o makespan. Consequentemente, as funções objetivo elaboradas trabalham na tentativa de normalizar ou adaptar valores referentes ao tempo de finalização do escalonamento.

Figura 89 – Funções objetivo em codificações baseadas em ordenação.

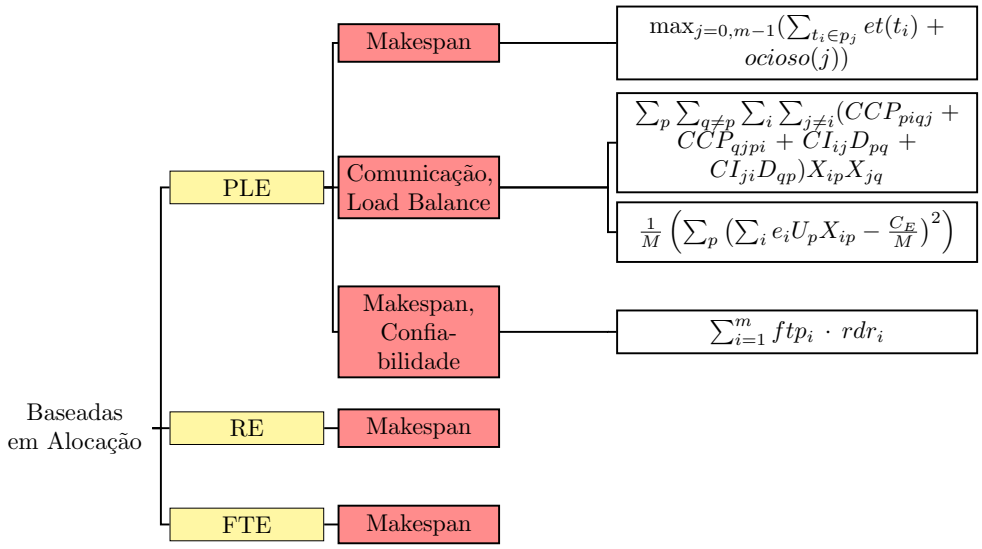


Fonte: do autor.

Em seguida, a Figura 90 distribui as funções objetivo nas codificações baseadas em alocação, sendo essas as representações PLE, RE e FTE. Ao contrário da categoria anterior, as codificações baseadas em alocação lidam com outras métricas além do makespan, sendo as métricas de custo de comunicação, load balance e confiabilidade também acrescentadas na avaliação. As codificações RE e FTE não têm suas funções objetivo apresentadas uma vez que estas não são descritas explicitamente nos trabalhos levantados.

A próxima categoria apresentada reúne as codificações que combinam alocação e ordenação, neste contexto, as representações TLE, ME, MSE e TE. Devido a quantidade de funções objetivo observadas, a síntese dessa categoria é dividida em duas figuras. Primeiramente, a Figura 91 aborda as funções objetivo das representações TLE e ME. Métricas como o makespan e custo energético são avaliadas na codificação TLE, enquanto que a ME, apesar de lidar com o makespan, não possui uma função objetivo descrita explici-

Figura 90 – Funções objetivo em codificações baseadas em alocação.

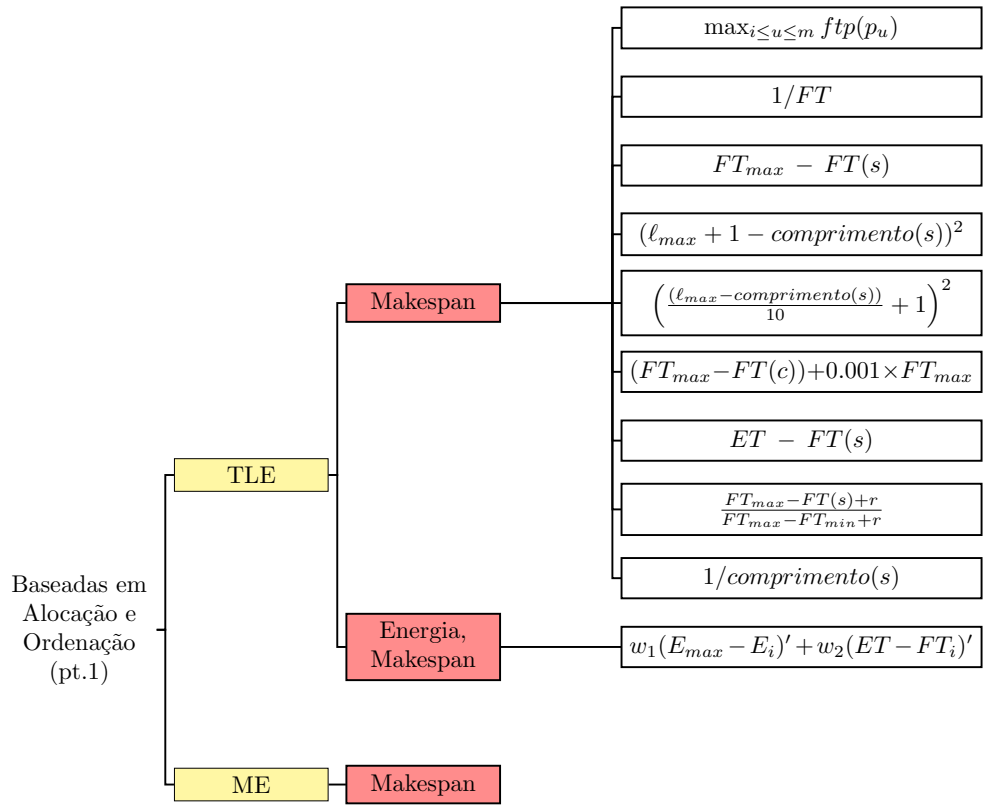


Fonte: do autor.

tamente. Complementando a categoria de funções objetivo em codificações baseadas em ordenação e alocação, a Figura 92 sintetiza as funções objetivo das demais codificações da categoria, sendo essas a MSE e a TE. Conforme resumido na figura, essa categoria de codificações também lida com métricas relacionadas ao load balance, flowtime, confiabilidade, quantidade de processadores, utilização de recursos e paralelização, sendo a maior variedade reunida na codificação MSE.

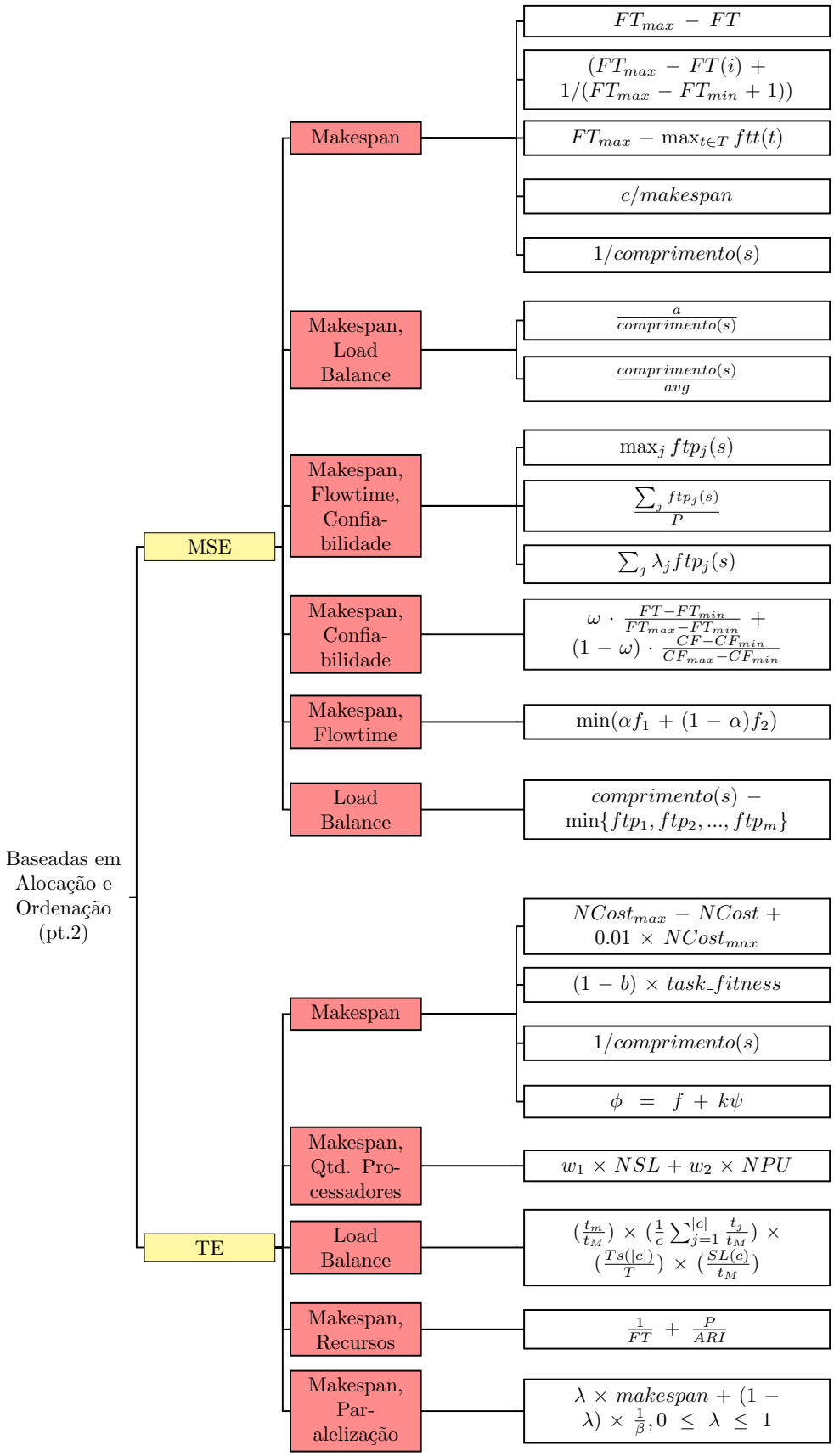
Por fim, a Figura 93 sintetiza as funções objetivo das demais codificações, desse modo, as representações MLE, DLE, DVFSE e QE. Conforme algumas codificações supracitadas, as representações MLE e QE não possuem suas funções objetivo nitidamente destacadas na coleção de trabalhos. Sobre as métricas, a codificação DVFSE reúne, além do makespan, custo energético e avaliação da temperatura. Naturalmente, essas funções objetivo/métricas, energia e temperatura, estão relacionadas às características estruturais da codificação DVFSE.

Figura 91 – Funções objetivo em codificações baseadas em alocação e ordenação (referente às codificações TLE e ME).



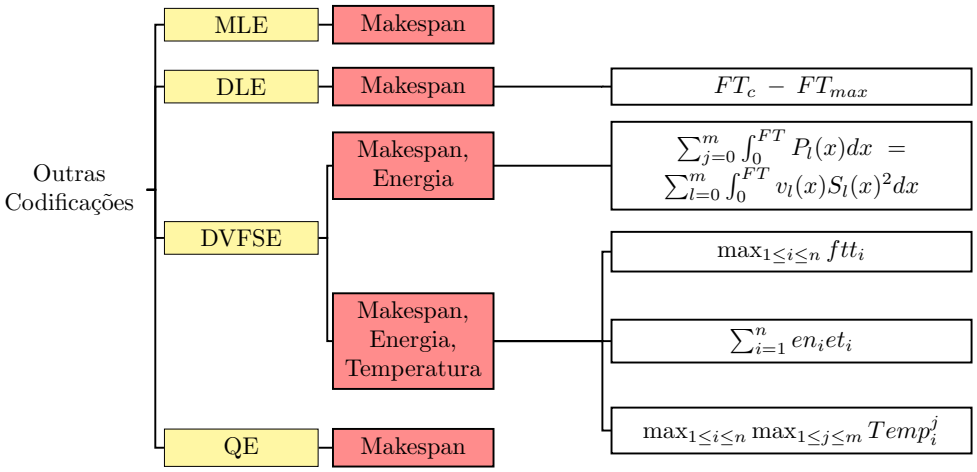
Fonte: do autor.

Figura 92 – Funções objetivo em codificações baseadas em alocação e ordenação (referente às codificações MSE e TE).



Fonte: do autor.

Figura 93 – Funções objetivo nas demais codificações.



Fonte: do autor.

Capítulo 6

Experimentos

Este capítulo apresenta uma análise das principais codificações levantadas. Desse modo, as codificações TLE, MSE, OLE e PLE foram analisadas sobre vários grafos de tarefas. As medidas de desempenho utilizadas foram o makespan, flowtime e load balance.

A Seção 6.1 introduz os principais aspectos considerados nos experimentos. Em seguida, uma análise de dispersão é apresentada na Seção 6.2.

6.1 Método

Guzek et al. (2014) destacam uma possível divisão nas diferentes codificações de cromossomos: representação por ordenação, representação por alocação e representação direta. Com base nessas classificações, o experimento apresentado aqui considerou cada grupo, tendo como base as representações diretas TLE e MSE (sendo essas as mais populares), a representação por ordenação, OLE, e a representação por alocação, PLE. O desempenho dessas codificações foi medido através das métricas de makespan, flowtime e load balance. Portanto, quatro codificações foram empregadas sobre três métricas de desempenho.

Dois experimentos diferentes foram executados, sendo o repositório STG (LABORATORY, 2001) o *benchmark* empregado. Em resumo, o primeiro experimento consiste em um teste que considera variados cenários com diversas quantidades de tarefas. Em cada um desses cenários, uma população de 1000 indivíduos é avaliada. Em contrapartida, o segundo experimento analisa diversas populações de 100 indivíduos em um único cenário, entretanto, com uma variação no valor da semente em cada uma das gerações dessas populações.

No primeiro experimento, considerando o repositório STG, o primeiro grafo de tarefas de cada subdiretório da versão 1 de testes foi selecionado.Consequentemente, o primeiro experimento utilizou grafos de 50, 100, 500, 700, 900, 1100, 1300, 1500, 1700, 1900, 2100, 2300, 2500 e 2700 tarefas. Conforme os arquivos do diretório STG, o custo de comunicação não foi considerado. Além disso, para cada grafo analisado, o experimento observou a

execução sobre um ambiente homogêneo com 2, 4 e 8 processadores. Neste cenário, para cada experimento, uma população de 1000 cromossomos foi gerada para codificação. Por fim, a aleatoriedade empregada em diversos processos do experimento foi comandado por valor de semente (*seed*), de forma que seja possível reproduzir os resultados aqui descritos. Um resumo das informações do primeiro experimento é apresentado na Tabela 10.

Tabela 10 – Características gerais do primeiro experimento.

Aspecto	**Descrição**
Quantidade de Tarefas	50, 100, 500, 700, 900, 1100, 1300, 1500, 1700, 1900, 2100, 2300, 2500 e 2700
Quantidade de Processadores	2, 4 e 8
Arquitetura	Homogênea
Custo de Comunicação	Não considerado
Métricas	Makespan, Flowtime e Load Balance
Codificações	TLE, MSE, OLE e PLE
Valor da Semente (*seed*)	1

O Segundo experimento também analisa as métricas do makespan, flowtime e load balance. A principal divergência quanto ao primeiro experimento é a geração da população. Enquanto o primeiro experimento utiliza um único valor de semente para todas as populações, o segundo experimento varia esse valor para cada população gerada. Além disso, o segundo experimento considera um único cenário com 2700 tarefas e 8 processadores disponíveis. Neste cenário, são geradas 10 populações de 100 indivíduos para cada codificação analisada, sendo que cada grupo populacional obedece um valor de semente diferente. Por exemplo, para a representação TLE, são geradas 10 populações a partir de um valor de semente. Em seguida, para cada representação, são geradas mais 10 populações diferentes. A relação entre cada grupo populacional do experimento e seus respectivos valores de semente é sintetizada na Tabela 11. Os valores de semente empregados foram gerados a partir da execução da rotina *clock* da *C time Library* (*time.h*) sobre o próprio código binário executável utilizado no experimento.

Para os dois experimentos executados, uma biblioteca de representações foi desenvolvida, sendo o código fonte escrito[1] em Linguagem C e executado em uma máquina hospedeira com especificações conforme a Tabela 12. Por fim, em ambos os experimentos, os testes foram executados seguindo uma ordem de lote (*batch*).

Conforme citado anteriormente, o experimento empregou quatro codificações diferentes: TLE, MSE, OLE e PLE. Em todas elas e em todos os experimentos, o mecanismo para garantir a geração de soluções válidas foi a função $altura'$ (Expressão 16) proposta por Hou, Ansari e Ren (1994). As codificações implementadas, TLE, MSE, OLE e PLE, seguiram os modelos de Hou, Ansari e Ren (1994), Wang et al. (1997), Kwok e Ahmad (1997) e Benten e Sait (1994), respectivamente.

[1] Disponível em: <github.com/anotherduardo/MTSP-Dispersion-Experiment.git>.

Tabela 11 – Valores de semente empregados em cada população analisada.

População	**Semente (*seed*)**
1	1472
2	1647
3	1335
4	1340
5	1551
6	1155
7	1367
8	1388
9	1425
10	1452

Tabela 12 – Especificações da máquina hospedeira.

Item	**Descrição**
SO	Debian 10 buster
Kernel	x86 64 Linux 4.19.0-5-amd64
CPU	Intel Core i3 M 370 @ 4x 2.399GHz
GPU	Mesa DRI Intel(R) Ironlake Mobile
RAM	3746 MiB

Na geração da população, todas as codificações foram concebidas aleatoriamente, isto é, genes responsáveis por ordenação ou alocação foram sorteados na início do algoritmo. Ao final, os resultados gerados foram analisados conforme a Seção 6.2.

6.2 Análise de Dispersão

Em conformidade às definições na Seção 6.1, os experimentos foram executados sobre as métricas de makespan, flowtime e load balance, sendo essas baseadas nos trabalhos de Hou, Ansari e Ren (1994), Kaur, Chhabra e Singh (2010a) e Omara e Arafa (2009), respectivamente.

A Subseção 6.2.1 apresenta uma análise dos resultados referentes ao primeiro experimento. Posteriormente, uma análise sobre os resultados do segundo experimento é apresentada na Subseção 6.2.2.

6.2.1 Experimento 1

A primeira métrica analisada foi o makespan. Sendo a métrica mais popular, o makespan avalia o tempo de finalização do escalonamento. A Tabela 13 reúne o valor médio ($\overline{x}$) de makespan para cada um dos casos de teste. Além disso, também são apresentados medidas quanto à variância (σ^2) e o desvio padrão (σ).

Naturalmente, quanto maior a quantidade de tarefas, maior será o valor do makespan. Também é possível observar um desempenho melhor da codificação OLE perante as demais. Isso se deve ao mecanismo utilizado na alocação dos processadores: Algoritmo de Minimização do Tempo de Inicialização (Algoritmo 3). Conforme citado anteriormente, a OLE foi implementada sobre o modelo de Kwok e Ahmad (1997). A OLE não mantêm uma lista de processadores alocados, de forma que seu escalonamento é gerado no momento da decodificação. Essa técnica melhora o desempenho na alocação, consequentemente, gerando melhores escalonamentos. Entretanto, devido o viés refletido na heurística, o mecanismo de alocação utilizado pela OLE tem impacto direto na redução da diversidade populacional, o que pode prejudicar o processo de busca de um GA por melhores soluções. Também é possível observar um equilíbrio no desempenho entre TLE, MSE e PLE. Essas codificações são bastante semelhantes, porém, com uma maior divergência na variedade de operadores genéticos. A conformidade no desempenho do makespan é resultado do processo de inicialização equivalente, independente da "forma" ou distribuição dos dados na codificação.

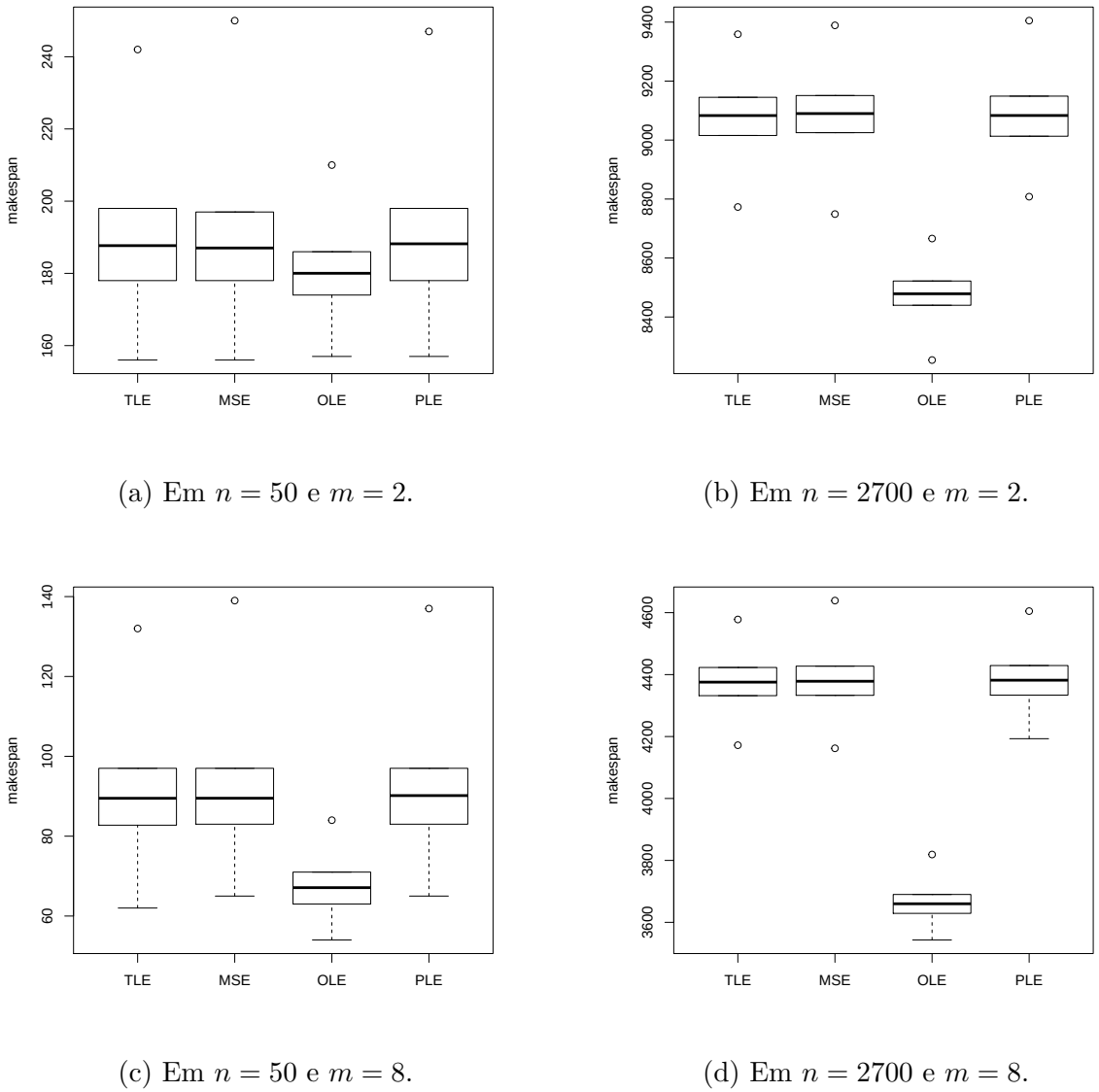
O diagrama de caixas apresentado na Figura 94 ilustra o comportamento dos resultados em alguns casos específicos. Nas Figuras 94(a) e 94(c), é apresentado a distribuição de soluções em um cenário com poucas tarefas ($n = 50$) e um ambiente com dois processadores ($m = 2$). Em contrapartida, nas Figuras 94(b) e 94(d), oito processadores ($m = 8$) são escalonados a uma quantidade maior de tarefas ($n = 2700$). Em ambos os casos, é possível observar o melhor desempenho da representação OLE, inclusive nos indivíduos com o maior valor de makespan. Porém, uma variedade maior de soluções é encontradas nas outras representações. Conforme a Tabela 13, a equivalência no desempenho das codificações TLE, MSE e PLE devido a forma de inicialização é apresentada.

A próxima métrica analisada é o flowtime. A Tabela 14 sumariza os resultados encontrados na medição do flowtime de forma análoga ao makespan. Para simplificação, a variância não foi exibida. A métrica do flowtime soma os tempos de finalização de todas as tarefas do escalonamento. Consequentemente, os valores exibidos possuem uma escala maior do que o makespan. Em resumo, conforme a análise anterior, a representação OLE apresenta um melhor desempenho. Além disso, novamente, as codificações TLE, MSE e PLE possuem resultados equivalentes.

Semelhantemente, o diagrama de caixas é apresentado na Figura 95 para verificar a distribuição dos indivíduos. Em conformidade ao resultado anterior, análise do makespan, a representação OLE possui melhores resultados e as demais codificações apresentam uma distribuição semelhante.

Por fim, uma análise sobre a métrica do load balance foi efetuada. Load balance é empregado na tentativa de avaliar a distribuição de tarefas dentre os processadores do conjunto. A Tabela 15 apresenta os resultados levantados sobre o load balance. A maior variação de resultados é encontrada quando não há uma quantidade muito extensa de

Figura 94 – Diagrama de caixa para análise do makespan.



(a) Em $n = 50$ e $m = 2$.

(b) Em $n = 2700$ e $m = 2$.

(c) Em $n = 50$ e $m = 8$.

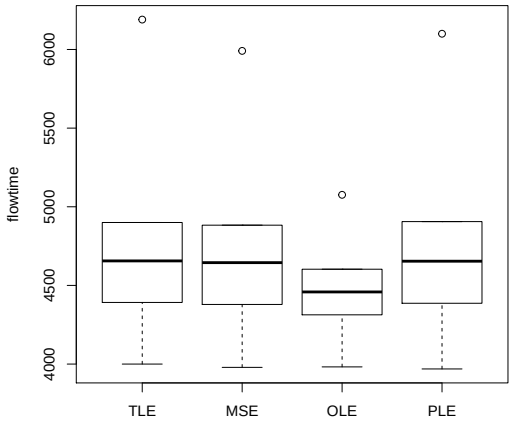
(d) Em $n = 2700$ e $m = 8$.

Fonte: do autor.

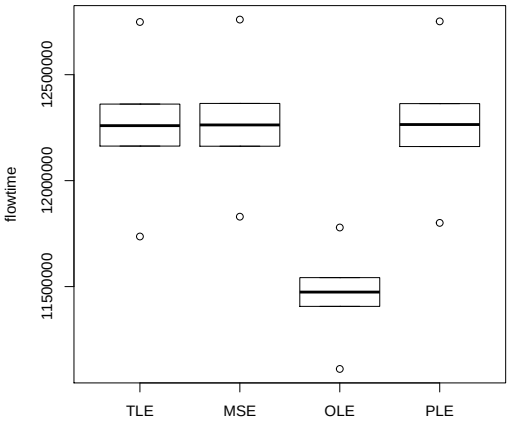
tarefas. Isso corresponde aos resultados semelhantes na maior parte das linhas da Tabela 15.

De forma análoga às análises anteriores, o diagrama de caixa é apresentado na Figura 96 para observar a distribuição dos resultados do load balance. Na Figura 15(b), é possível observar uma distribuição igual entre os processadores. Em contrapartida, a Figura 15(c) apresenta uma situação com uma variedade maior de processadores e menos tarefas para distribuição. Conforme os resultados anteriores, a codificação OLE também apresenta um melhor desempenho no load balance. Assim como nas outras métricas, isso se deve ao algoritmo de distribuição de tarefas implementado nesse modelo. Similarmente, os resultados apontam a equivalência entre TLE, MSE, e PLE.

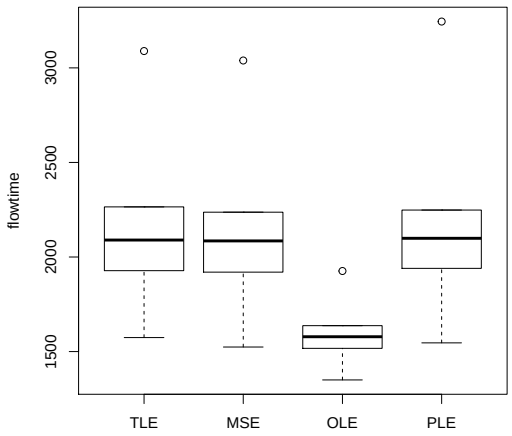
Figura 95 – Diagrama de caixa para análise do flowtime.



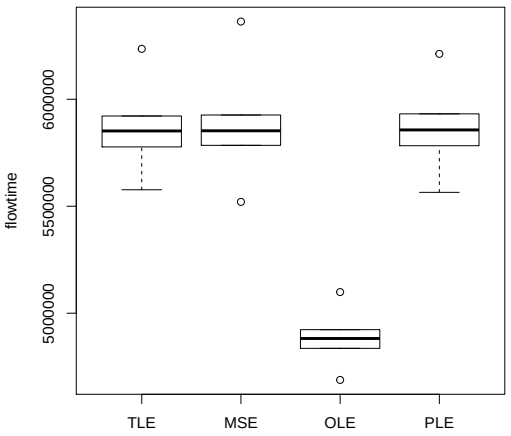
(a) Em $n = 50$ e $m = 2$.



(b) Em $n = 2700$ e $m = 2$.



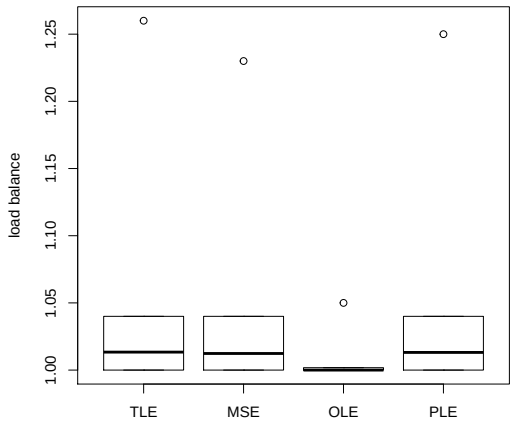
(c) Em $n = 50$ e $m = 8$.



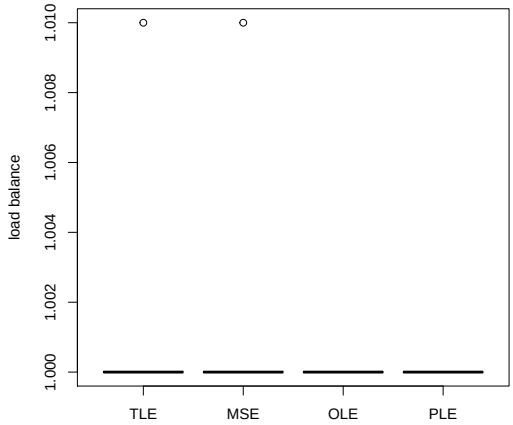
(d) Em $n = 2700$ e $m = 8$.

Fonte: do autor.

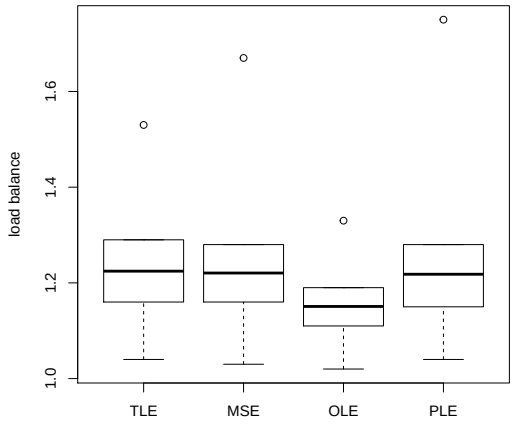
Figura 96 – Diagrama de caixa para análise do load balance.



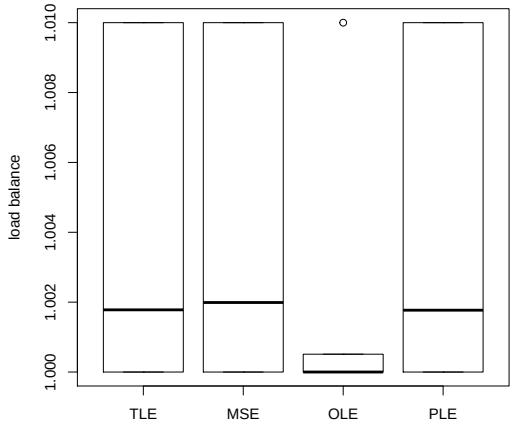
(a) Em $n = 50$ e $m = 2$.



(b) Em $n = 2700$ e $m = 2$.



(c) Em $n = 50$ e $m = 8$.



(d) Em $n = 2700$ e $m = 8$.

Fonte: do autor.

Tabela 13 – Análise do Makespan.

n	m	TLE			MSE			OLE			PLE		
		$\overline{x}$	σ^2	σ	$\overline{x}$	σ^2	σ	$\overline{x}$	σ^2	σ	$\overline{x}$	σ^2	σ
50	2	188.37	206.55	14.37	188.06	219.13	14.80	180.05	71.07	8.43	188.38	209.17	14.46
100	2	335.28	333.24	18.25	335.23	332.97	18.25	307.20	110.03	10.49	335.48	351.35	18.74
500	2	1706.08	1719.40	41.47	1710.02	1939.60	44.04	1578.01	747.42	27.34	1708.50	1794.83	42.37
700	2	2371.49	2345.09	48.43	2375.40	2557.43	50.57	2170.32	749.87	27.38	2372.46	2353.29	48.51
900	2	2977.67	2845.98	53.35	2979.35	2936.85	54.19	2806.76	1291.52	35.94	2977.49	3138.74	56.02
1100	2	3701.63	3745.71	61.20	3708.22	3724.55	61.03	3443.92	1550.59	39.38	3702.58	4229.62	65.04
1300	2	4477.13	4845.76	69.61	4476.06	5206.94	72.16	4127.05	1732.41	41.62	4472.82	4699.47	68.55
1500	2	5003.40	5092.27	71.36	5006.74	5297.30	72.78	4618.63	1766.50	42.03	5001.66	5465.04	73.93
1700	2	5724.46	6084.15	78.00	5727.17	5984.54	77.36	5324.61	2328.66	48.26	5722.78	5879.92	76.68
1900	2	6446.80	6585.98	81.15	6453.10	6588.18	81.17	5991.06	2576.24	50.76	6445.81	6314.26	79.46
2100	2	7033.25	7028.14	83.83	7032.26	7976.18	89.31	6600.28	2861.80	53.50	7034.54	7043.73	83.93
2300	2	7676.33	7931.01	89.06	7673.92	8541.79	92.42	7219.36	3355.61	57.93	7672.88	8243.31	90.79
2500	2	8405.62	8898.17	94.33	8406.40	8755.49	93.57	7813.19	3556.62	59.64	8405.66	8859.56	94.13
2700	2	9082.13	9461.43	97.27	9087.73	8897.88	94.33	8479.43	3803.76	61.67	9084.57	9180.39	95.81
50	4	123.83	174.87	13.22	124.32	174.90	13.23	102.61	37.60	6.13	124.42	163.41	12.78
100	4	232.74	262.68	16.21	232.43	265.40	16.29	200.94	93.44	9.67	232.59	275.53	16.60
500	4	1135.12	1492.57	38.63	1138.50	1506.89	38.82	988.20	566.57	23.80	1136.58	1521.27	39.00
700	4	1570.14	2068.61	45.48	1573.66	2093.36	45.75	1347.02	726.02	26.94	1569.57	2000.27	44.72
900	4	1952.53	2471.90	49.72	1954.76	2573.62	50.73	1694.21	997.51	31.58	1954.91	2435.34	49.35
1100	4	2471.88	3339.64	57.79	2475.32	3322.33	57.64	2126.34	1149.73	33.91	2471.40	3226.97	56.81
1300	4	2972.30	3778.24	61.47	2977.27	3696.99	60.80	2575.49	1516.17	38.94	2972.23	3966.96	62.98
1500	4	3315.88	4118.34	64.17	3318.18	4428.41	66.55	2860.80	1608.68	40.11	3315.08	4327.80	65.79
1700	4	3788.06	5117.88	71.54	3788.87	4943.39	70.31	3286.93	1716.57	41.43	3782.83	4961.08	70.43
1900	4	4285.62	5626.09	75.01	4287.82	5375.74	73.32	3752.75	2150.08	46.37	4284.03	5534.21	74.39
2100	4	4677.40	5903.92	76.84	4678.88	5691.71	75.44	4079.09	2380.10	48.79	4679.99	6171.25	78.56
2300	4	5051.05	6351.97	79.70	5049.89	6355.37	79.72	4393.32	2668.61	51.66	5048.93	6614.12	81.33
2500	4	5536.29	7296.16	85.42	5535.46	7289.71	85.38	4817.71	2826.75	53.17	5537.80	7137.25	84.48
2700	4	5988.94	7162.44	84.63	5999.35	7185.97	84.77	5145.29	2946.22	54.28	5996.39	7367.47	85.83
50	8	89.98	106.22	10.31	90.01	115.58	10.75	67.22	27.29	5.22	90.33	98.38	9.92
100	8	179.63	172.52	13.13	179.90	168.16	12.97	158.29	68.75	8.29	179.41	181.13	13.46
500	8	834.39	967.03	31.10	837.52	938.57	30.64	712.79	376.13	19.39	836.63	926.31	30.44
700	8	1152.43	1227.83	35.04	1154.74	1341.30	36.62	963.88	441.03	21.00	1153.57	1272.39	35.67
900	8	1411.91	1713.82	41.40	1412.35	1734.40	41.65	1186.19	651.65	25.53	1413.77	1749.19	41.82
1100	8	1823.61	2173.85	46.62	1824.77	2285.41	47.81	1522.68	847.30	29.11	1820.56	2293.69	47.89
1300	8	2188.86	2667.73	51.65	2191.39	2665.97	51.63	1874.71	1130.18	33.62	2187.81	2600.44	50.99
1500	8	2432.22	2744.39	52.39	2431.14	2878.91	53.66	2053.55	1130.64	33.63	2429.16	2749.94	52.44
1700	8	2770.37	3327.13	57.68	2769.03	3375.58	58.10	2338.84	1233.37	35.12	2769.02	3235.83	56.88
1900	8	3153.81	3697.10	60.80	3157.72	3404.56	58.35	2684.44	1612.65	40.16	3153.41	3616.01	60.13
2100	8	3447.79	3667.59	60.56	3446.91	3565.51	59.71	2950.57	1655.05	40.68	3448.93	4118.48	64.18
2300	8	3661.36	4386.56	66.23	3664.64	4450.08	66.71	3071.18	1538.25	39.22	3662.89	4161.36	64.51
2500	8	4021.32	4795.98	69.25	4017.80	4717.20	68.68	3343.10	1773.32	42.11	4021.26	4467.94	66.84
2700	8	4376.33	4720.99	68.71	4380.42	5096.44	71.39	3660.34	1933.72	43.97	4381.80	4927.03	70.19

Tabela 14 – Análise do Flowtime.

n	m	TLE		MSE		OLE		PLE	
		$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
50	2	4678.05	373.24	4661.18	372.74	4462.05	204.01	4667.88	366.17
100	2	15916.24	956.73	15943.15	1005.94	14848.28	592.39	15928.83	967.02
500	2	413882.81	11214.62	414724.13	12028.25	381806.00	7222.18	414460.21	11885.85
700	2	824117.76	19724.84	825725.47	19771.92	754774.55	10908.82	824341.42	19565.33
900	2	1337208.77	28950.37	1337041.83	28469.87	1259246.93	18946.36	1337604.77	29524.51
1100	2	2025790.98	37213.66	2029408.88	38878.48	1896001.39	25352.80	2025759.90	40192.68
1300	2	2888749.81	53884.95	2889676.45	52630.84	2670926.67	31361.47	2887667.71	50777.49
1500	2	3712105.58	62113.30	3715857.36	61365.06	3436035.44	36345.43	3711605.14	61833.10
1700	2	4870907.51	75428.15	4877202.17	77132.90	4533156.13	47317.33	4871785.45	74224.38
1900	2	6083580.23	89697.82	6091225.48	87765.44	5655254.16	54079.79	6081343.85	85598.26
2100	2	7376850.55	101932.85	7377569.54	107184.40	6934728.16	65795.77	7381831.75	106534.98
2300	2	8821216.48	118832.25	8817666.90	119270.77	8312548.06	77917.06	8821138.01	122836.47
2500	2	10573655.60	136272.83	10574585.99	132666.21	9806803.77	85314.33	10573241.91	134546.37
2700	2	12261580.76	149122.52	12264188.40	144800.62	11474746.94	98472.63	12264305.18	151058.24
50	4	2991.65	325.96	2995.41	316.14	2492.69	135.88	3005.82	320.06
100	4	10717.13	848.65	10727.31	854.50	9515.62	513.40	10699.58	858.32
500	4	274548.94	10308.62	275317.26	10586.98	242097.30	7109.90	274772.87	10820.49
700	4	542349.98	18537.38	543960.52	18312.68	462799.29	10476.04	542045.61	18031.32
900	4	874854.76	25605.73	876041.31	26550.45	762446.32	16666.84	876225.42	25462.14
1100	4	1348039.11	36233.48	1348770.85	36983.89	1161753.52	21582.57	1346856.74	35514.40
1300	4	1916988.21	46600.13	1920627.87	44278.47	1667944.36	28008.11	1918342.39	46929.71
1500	4	2444363.29	54741.61	2445533.93	56649.82	2118856.64	34555.32	2444753.65	56178.76
1700	4	3218261.75	68995.38	3220263.51	69061.92	2805063.59	41080.09	3215577.33	71100.15
1900	4	4042845.34	80359.60	4046199.51	80835.85	3546359.23	50839.95	4041952.55	80312.68
2100	4	4893002.05	95382.76	4894987.67	93917.39	4255616.58	58386.68	4897100.18	93063.20
2300	4	5810044.17	105946.69	5807318.79	106155.38	5028592.01	66050.86	5810868.10	107793.00
2500	4	6976210.57	126558.60	6978244.50	122756.29	6061355.83	76209.61	6980933.82	119720.35
2700	4	8050670.48	131680.75	8062976.92	132919.44	6909906.27	83674.25	8060081.76	133124.26
50	8	2104.36	246.57	2097.22	238.17	1581.83	93.89	2111.29	236.39
100	8	7984.81	639.92	8020.00	656.91	7072.42	381.74	7980.81	638.76
500	8	201505.59	8352.56	202247.01	8315.31	174494.66	5456.73	201698.63	8370.96
700	8	394980.99	13928.57	395716.57	14461.46	327477.98	7970.32	395097.13	14070.30
900	8	629512.48	21890.01	629991.49	21608.61	530369.85	12715.70	631467.98	21638.55
1100	8	988562.64	29357.34	986600.69	30150.23	819147.19	17753.94	985376.12	30153.30
1300	8	1413035.00	36780.33	1415306.34	37709.46	1216430.66	24519.24	1411642.59	37540.20
1500	8	1781268.25	44065.83	1780421.24	46057.35	1503292.70	28763.08	1779825.88	44590.07
1700	8	2346682.48	56261.97	2346100.98	56487.82	1987551.77	35118.79	2345856.86	56373.25
1900	8	2974685.08	66343.85	2978432.71	64247.34	2532292.48	42468.55	2974902.17	64605.87
2100	8	3599186.63	74711.29	3596501.37	72510.52	3069996.28	47318.42	3600376.06	77146.56
2300	8	4214216.23	87158.31	4218841.51	87872.05	3550478.10	54312.34	4218974.92	85583.09
2500	8	5083387.52	99593.81	5079789.06	96160.19	4238681.88	58749.58	5086420.83	95585.54
2700	8	5852237.67	106084.30	5854069.83	109923.91	4881044.59	67193.76	5856011.38	109259.86

Tabela 15 – Análise do Load Balance.

n	m	TLE			MSE			OLE			PLE		
		$\overline{x}$	σ^2	σ	$\overline{x}$	σ^2	σ	$\overline{x}$	σ^2	σ	$\overline{x}$	σ^2	σ
50	2	1.03	0.00	0.04	1.02	0.00	0.04	1.00	0.00	0.01	1.03	0.00	0.04
100	2	1.01	0.00	0.02	1.01	0.00	0.02	1.00	0.00	0.01	1.01	0.00	0.02
500	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
700	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
900	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1100	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1300	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1500	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1700	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1900	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2100	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2300	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2500	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2700	2	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
50	4	1.09	0.00	0.06	1.10	0.00	0.06	1.07	0.00	0.04	1.09	0.00	0.06
100	4	1.05	0.00	0.03	1.05	0.00	0.03	1.01	0.00	0.01	1.05	0.00	0.03
500	4	1.01	0.00	0.01	1.01	0.00	0.01	1.01	0.00	0.01	1.01	0.00	0.01
700	4	1.01	0.00	0.01	1.01	0.00	0.01	1.01	0.00	0.01	1.01	0.00	0.01
900	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1100	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1300	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1500	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1700	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1900	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2100	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2300	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2500	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2700	4	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
50	8	1.23	0.01	0.09	1.23	0.01	0.09	1.15	0.00	0.06	1.23	0.01	0.10
100	8	1.12	0.00	0.04	1.12	0.00	0.04	1.08	0.00	0.02	1.12	0.00	0.04
500	8	1.02	0.00	0.01	1.02	0.00	0.01	1.01	0.00	0.00	1.02	0.00	0.01
700	8	1.01	0.00	0.01	1.01	0.00	0.01	1.01	0.00	0.00	1.01	0.00	0.01
900	8	1.01	0.00	0.00	1.01	0.00	0.00	1.01	0.00	0.00	1.01	0.00	0.00
1100	8	1.01	0.00	0.00	1.01	0.00	0.00	1.01	0.00	0.00	1.01	0.00	0.00
1300	8	1.01	0.00	0.00	1.01	0.00	0.00	1.00	0.00	0.00	1.01	0.00	0.00
1500	8	1.01	0.00	0.00	1.01	0.00	0.00	1.00	0.00	0.00	1.01	0.00	0.00
1700	8	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
1900	8	1.01	0.00	0.00	1.01	0.00	0.00	1.00	0.00	0.00	1.01	0.00	0.00
2100	8	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2300	8	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2500	8	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
2700	8	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00

6.2.2 Experimento 2

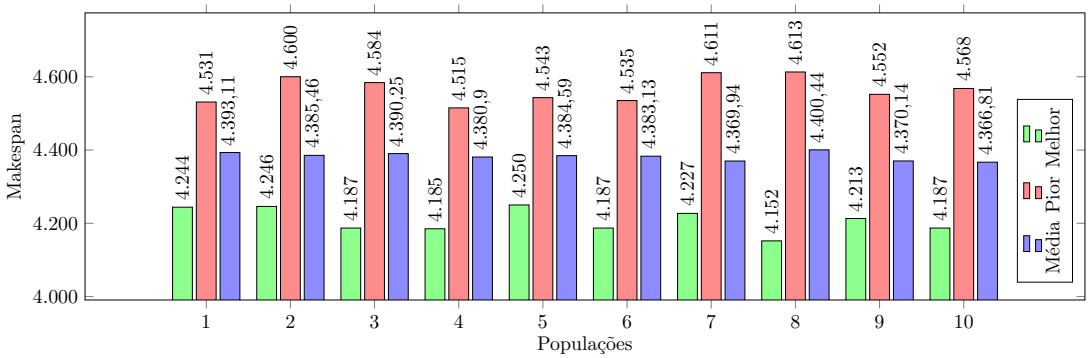
Inicialmente, a Figura 97 distribui a média de makespan para cada uma das dez populações geradas nas codificações analisadas. Sequencialmente, a figura distribui os dados sobre as representações TLE (Figura 97(a)), MSE (Figura 97(b)), OLE (Figura 97(c)) e PLE (Figura 97(d)). Em adição, a figura também exibe, para cada população, o melhor e o pior indivíduo encontrado. Os resultados da métrica do makespan são análogos ao experimento anterior, de forma que a OLE obteve indivíduos com o melhor desempenho. O desempenho da OLE também é refletido nos indivíduos com os mais altos valores de makespan em cada uma das populações. Esses indivíduos, apesar de apresentarem o pior desempenho dentro de suas respectivas populações, apresentam um

desempenho melhor do que os melhores indivíduos das demais representações. Além disso, novamente, as representações TLE, MSE e PLE apresentam desempenho semelhante. Cabe ressaltar, conforme supracitado, que apesar de apresentar um desempenho melhor, a codificação OLE utiliza uma heurística de alocação que pode tendenciar a geração de indivíduos, o que pode impactar na diversidade da população e, consequentemente, na busca por soluções melhores.

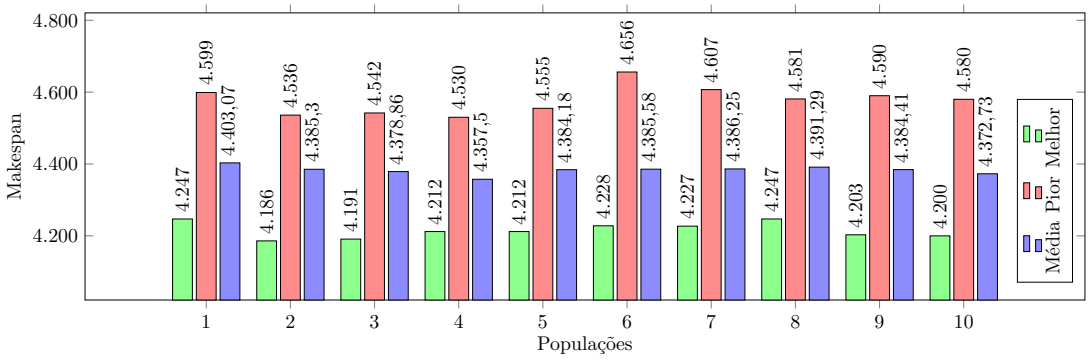
A próxima métrica analisada é o flowtime, sendo o desempenho das populações de cada representação apresentadas na Figura 98. Para critérios de ilustração e escala, os valores de flowtime foram divididos por 1000. O comportamento das populações e codificações é análogo ao emprego da métrica de makespan, ou seja, ao analisar o desempenho observando a métrica de flowtime, a representação OLE (Figura 98(c)) possui melhor desempenho e as demais representações apresentam um desempenho equivalente. Da mesma forma, os piores indivíduos (ao considerar o flowtime) na OLE possuem melhor desempenho do que os melhores indivíduos das outras representações e populações.

Por fim, a última métrica analisada é o load balance, sendo seus dados distribuídos na Figura 99. Dentre os gráficos ilustrados pela figura, é possível observar que todas as representações apresentaram uma baixa variabilidade nas avaliações. Isso é relacionado à natureza de avaliação do load balance neste cenário, ou seja, os escalonamentos tendem a ser balanceados devido a quantidade significativa de tarefas para serem distribuídas em poucas unidades (2700 tarefas para 8 processadores). Contudo, mesmo com a baixa variação na medição, a representação OLE apresenta o melhor desempenho, apesar de possuir alguns dos indivíduos de pior desempenho semelhantes aos piores indivíduos das demais populações (populações 5 e 7).

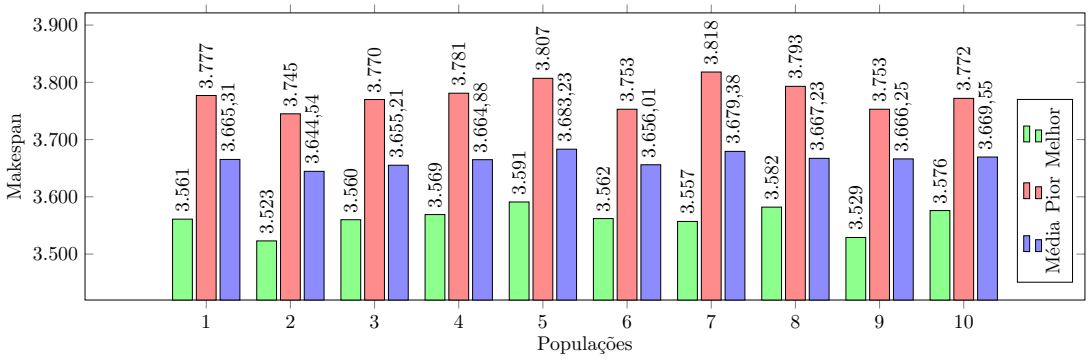
Figura 97 – Relação entre a métrica de makespan e as 10 populações geradas em cada representação.



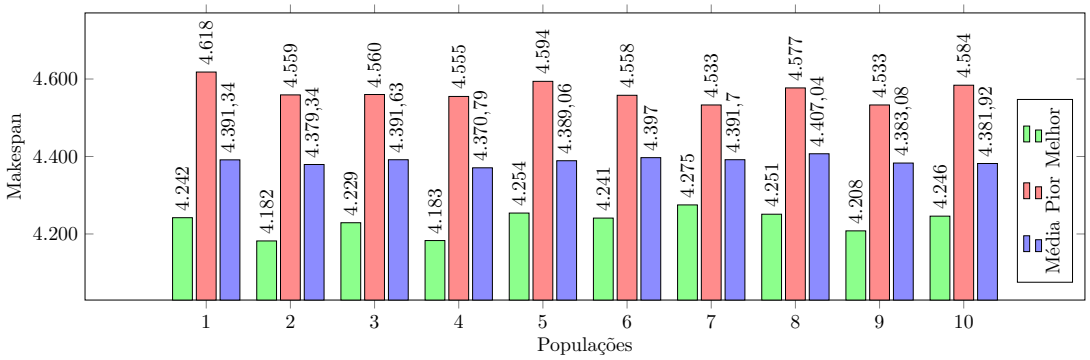
(a) Representação TLE.



(b) Representação MSE.



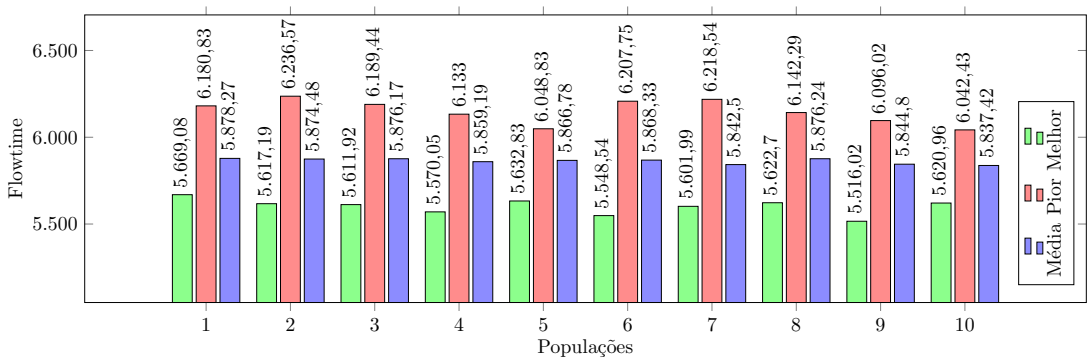
(c) Representação OLE.



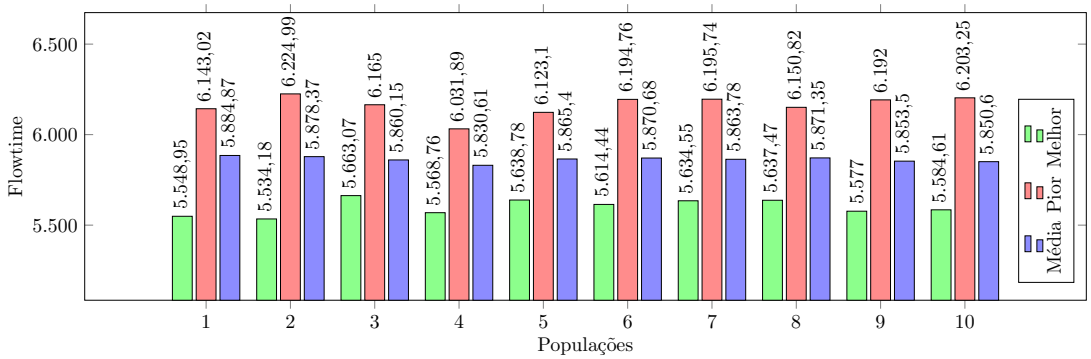
(d) Representação PLE.

Fonte: do autor.

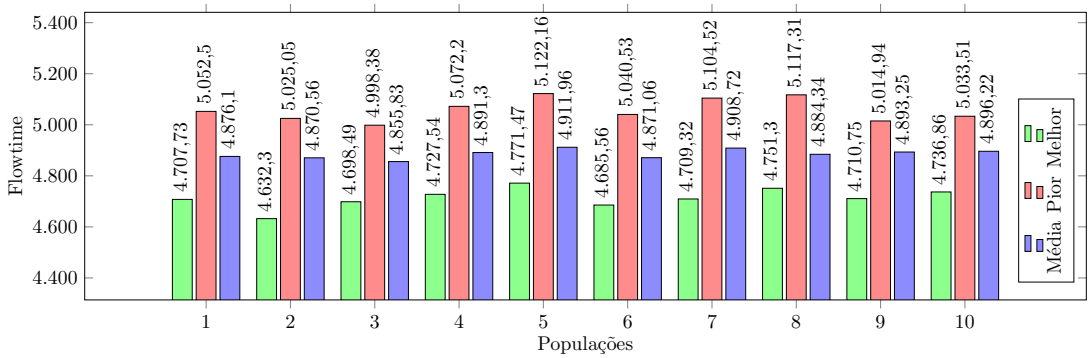
Figura 98 – Relação entre a métrica de flowtime e as 10 populações geradas em cada representação.



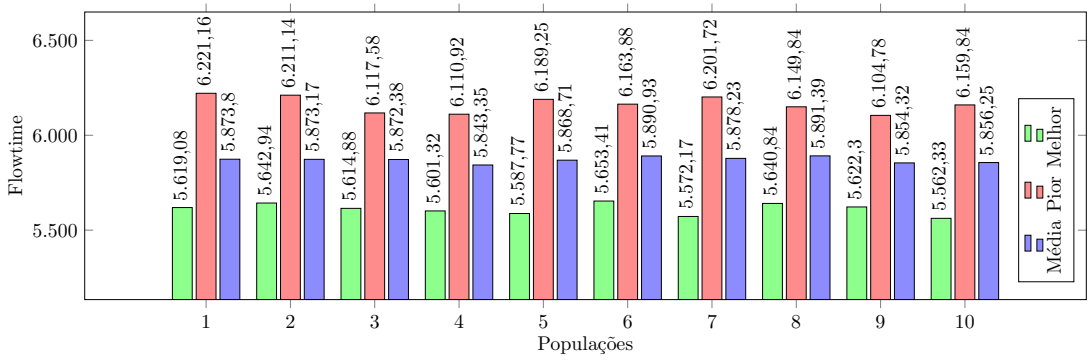
(a) Representação TLE.



(b) Representação MSE.



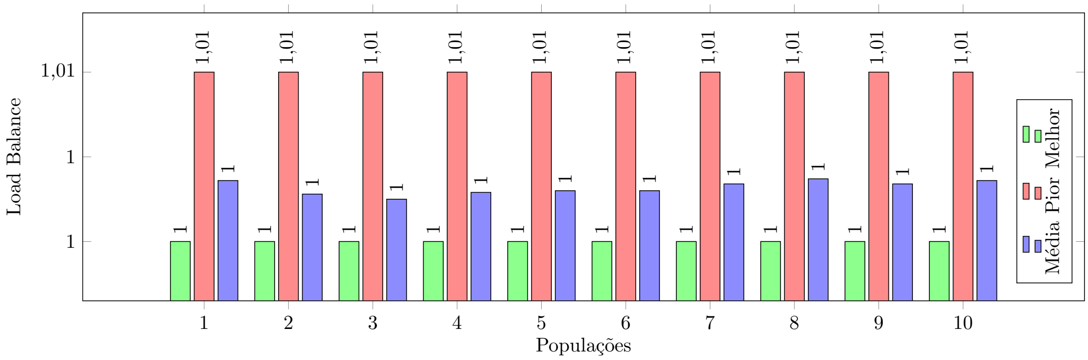
(c) Representação OLE.



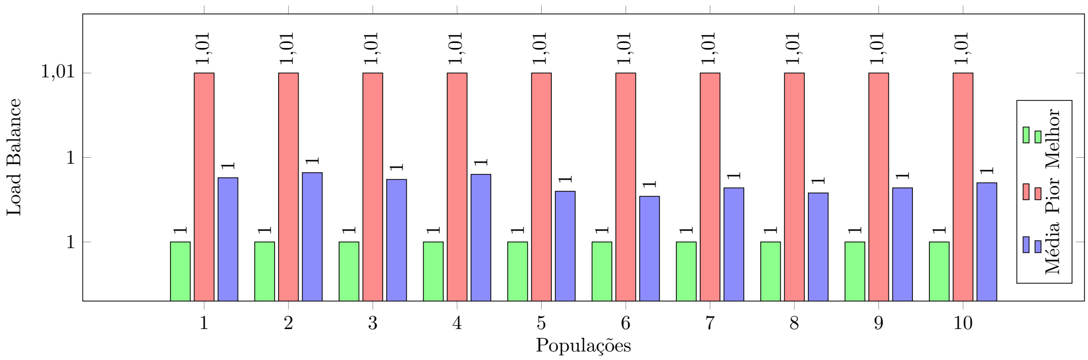
(d) Representação PLE.

Fonte: do autor.

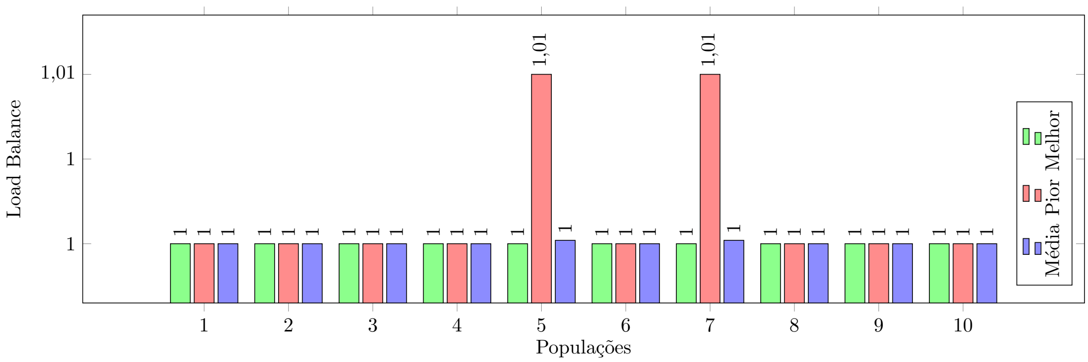
Figura 99 – Relação entre a métrica de load balance e as 10 populações geradas em cada representação.



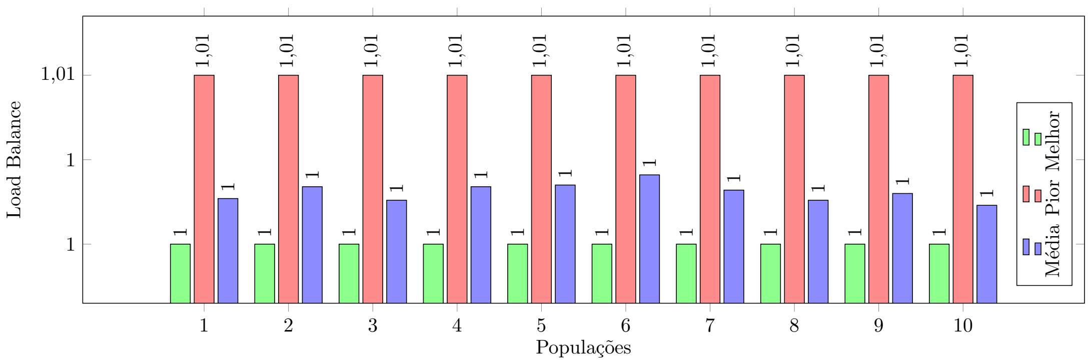
(a) Representação TLE.



(b) Representação MSE.



(c) Representação OLE.



(d) Representação PLE.

Fonte: do autor.

Capítulo 7

Conclusão

A aplicação de SLR é um processo eficiente para pesquisa, estudo e filtragem de trabalhos relacionados a uma questão de pesquisa. Seus métodos rigorosos permitem a execução de uma revisão não enviesada, de forma que a pesquisa foi conduzida inteiramente pelo protocolo de revisão. Desse modo, os 63 estudos selecionados foram justificados pelos critérios definidos no início do projeto.

Com a coleção de estudos formadas na SLR, foi possível observar os diversos esforços dos pesquisadores em melhor adaptar solução ao MTSP estático com o uso de GAs. Delimitando a investigação entre os períodos de 1990 a 2018, foram encontrados estudos nos mais diversos cenários, desde a aplicação do MTSP a um sistema homogêneo de processadores, até mesmo propostas de aplicação em sistemas heterogêneos baseados no controle de frequência. Além disso, foi possível observar a quantidade de estudos que lidam com o tratamento do envio de dados entre as tarefas, sendo essa uma característica a agravar a dificuldade na resolução.

A definição da coleção de estudos também foi essencial na extração das características dos GAs aplicados nessas soluções. Baseando-se na forma como os genes são organizados e como os operadores são aplicados, foram identificados 13 modelos de representações. Dentre esses, a codificação de alocação e escalonamento ($\approx 27.94\%$) e a codificação de lista topológica ($\approx 25\%$) são as abordagens mais utilizadas. Naturalmente, devido a popularidade nos estudos levantados, TLE e MSE são as representações com a maior variedade de operadores genéticos.

Além de contabilizar, a extração também foi aplicada no levantamento e sumarização das características dos operadores genéticos empregados. Aproximadamente, considerando cada cenário de codificação, foram identificados 39 formas de aplicar recombinações. Semelhantemente, foram levantados 39 modelos de mutação. Portanto, dada as particularidades das 13 codificações levantadas, 78 operadores genéticos foram encontrados e sumarizados. Neste levantamento, recombinações baseadas em um único ponto de corte são os modelos predominantes, enquanto que nas mutações, métodos baseados em bit-flip são as operações mais populares.

Também, dentre os elementos dos GAs, os procedimentos de seleção foram observados. O método de seleção por Roleta ($\approx 60\%$) é o mais utilizado, sendo seguido pelo método de Torneio ($\approx 16.16\%$). Nas estratégias elitistas, conduzir uma parcela da população para a próxima geração ($\approx 70\%$) e substituir cromossomos da população atual ($\approx 20\%$) são os modelos mais empregados.

No contexto do MTSP, a execução da SLR encontrou diversas métricas utilizadas para medir a qualidade das soluções. Quase unanimidade entre os trabalhos da coleção, a métrica do makespan é a medida mais empregada. Porém, estudos recentes apontam para uma ascensão na utilização de métricas voltadas a sustentabilidade, por exemplo, as métricas de temperatura dos processadores e custo energético.

Quanto ao cenário de aplicação, estando em 32 trabalhos da coleção ($\approx 50,79\%$), a arquitetura homogênea é o ambiente mais utilizado na coleção. O ambiente com processadores heterogêneos foi identificado em 25 trabalhos da coleção ($\approx 39,68\%$). Além disso, o custo de comunicação é abordado na maior parte dos trabalhos levantados ($\approx 79,36\%$).

Ao analisar a dispersão de soluções no experimento, foi possível observar a equivalência entre as codificações TLE, MSE e PLE na geração da população inicial. Em termos de desempenho, a codificação OLE obteve melhores resultados. Todavia, comparada às outras representações testadas, OLE utilizou um algoritmo eficiente para a alocação das tarefas, enquanto que as demais codificações partiram de alocações totalmente aleatórias. O algoritmo de alocação na OLE é necessário uma vez que a representação não armazena a relação de alocações do escalonamento, isto é, as alocações são efetuadas na decodificação. Além disso, a OLE traz pouca diversidade na população inicial, podendo tornar o processo de busca menos eficiente.

Em resumo, além da análise do comportamento de quatro codificações sobre três métricas de desempenho, essa dissertação contribuiu com a construção de uma síntese abrangente sobre diversos aspectos dos GAs aplicados ao MTSP. Com o intuito de responder às questões de pesquisa, os dados foram extraídos a partir de estudos reunidos por meio de uma seleção criteriosa e não tendenciosa. Desse modo, a organização de dados estabelecida nesta dissertação, seja a reunião dos estudos ou das caraterísticas das soluções, pode ser significativa na tomada de decisão e na realização de novas pesquisas sobre o tema.

Como sugestão inicial de trabalhos futuros, os experimentos podem ser estendidos para analisar possíveis correlações entre as métricas, ou seja, é possível observar se determinadas métricas podem influenciar diretamente no resultado das demais. Essa análise pode ser aplicada em métricas geralmente não empregadas na literatura, tais como a maximização da paralelização ou minimização do custo de comunicação, ou em novas métricas, como a minimização do consumo elétrico ou a minimização da temperatura do sistema. Desta forma, seria possível identificar métricas conflitantes, o que poderia contribuir na modelagem do problema em otimizações multiobjetivo.

Uma sugestão adicional é a comparação das representações em outros cenários com novas limitações. Por exemplo, cenários com processadores heterogêneos e com grafos de tarefas que considerem o custo de comunicação. A arquitetura destes cenários também poderiam ser expandida para modelos com controle de frequência de operação dos processadores, como o DVFS ou outra forma de manipular a velocidade de processamento. Com o emprego do custo de comunicação, novos experimentos também poderiam considerar cenários onde a duplicação de tarefas é aplicada, sendo esta uma técnica que expande a variabilidade de soluções. Uma nova comparação das representações sobre esses contextos poderia encontrar situações onde uma determinada forma de representação pode levar vantagem, dependendo da métrica de desempenho, perante os demais modelos levantados.

Por fim, uma outra sugestão de trabalhos futuros é a possibilidade da elaboração de novos modelos. Utilizando as características levantadas nessa dissertação, novas representações podem ser construídas. Esses novos modelos podem ser planejados para cenários específicos ao combinar elementos distintos dentre as representações. Consequentemente, essas novas codificações podem ser melhor adaptadas a estes cenários. Além disso, operadores genéticos também podem ser explorados para a geração desses novos modelos representativos.

Referências

AGUILAR, J.; GELENBE, E. Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences*, v. 97, n. 1-2, p. 199–219, 1997. Disponível em: <https://doi.org/10.1016/S0020-0255(96)00178-8>.

AHMAD, S. G.; LIEW, C. S.; MUNIR, E. U.; ANG, T. F.; KHAN, S. U. A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, v. 87, p. 80–90, 2016. Disponível em: <https://doi.org/10.1016/j.jpdc.2015.10.001>.

AHMAD, S. G.; MUNIR, E. U.; NISAR, W. PEGA: A performance effective genetic algorithm for task scheduling in heterogeneous systems. In: IEEE. *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems*. Liverpool, 2012. p. 1082–1087. Disponível em: <https://doi.org/10.1109/HPCC.2012.158>.

AKBARI, M. An efficient genetic algorithm for task scheduling on heterogeneous computing systems based on triz. *Journal of Advances in Computer Research*, v. 9, n. 3, p. 103–132, 2018.

AKBARI, M.; RASHIDI, H.; ALIZADEH, S. H. An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems. *Engineering Applications of Artificial Intelligence*, v. 61, p. 35–46, 2017. Disponível em: <https://doi.org/10.1016/j.engappai.2017.02.013>.

AMALARETHINAM, D. I. G.; SELVI, F. K. M. Grid scheduling strategy using GA (GSSGA). *International Journal of Computer Technology and Applications*, v. 6, n. 5, p. 1800–1806, 2012.

AMIRJANOV, A.; SOBOLEV, K. Scheduling of directed acyclic graphs by a genetic algorithm with a repairing mechanism. *Concurrency and Computation: Practice and Experience*, v. 29, n. 5, p. 1–10, 2017. Disponível em: <https://doi.org/10.1002/cpe.3954>.

AMPATZOGLOU, A.; STAMELOS, I. Software engineering research for computer games: A systematic review. *Information & Software Technology*, v. 52, n. 9, p. 888–901, 2010. Disponível em: <https://doi.org/10.1016/j.infsof.2010.05.004>.

AWADALL, M.; AHMAD, A.; AL-BUSAIDI, S. Min-min GA based task scheduling in multiprocessor systems. *International Journal of Engineering and Advanced Technology*, v. 2, n. 2, p. 2249–8958, 2013.

AZGHADI, M. R.; BONYADI, M. R.; HASHEMI, S.; MOGHADAM, M. E. A hybrid multiprocessor task scheduling method based on immune genetic algorithm. In: *Proceedings of the 5th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. Hong Kong: [s.n.], 2008. p. 1–4. Disponível em: <https://doi.org/10.4108/ICST.QSHINE2008.4263>.

BACK, T.; FOGEL, D.; MICHALEWICZ, Z. *Evolutionary Computation 2: Advanced Algorithms and Operators*. [S.l.]: Taylor & Francis, 2000. (Evolutionary computation). Disponível em: <https://doi.org/10.1201/9781420034349>. ISBN 9781420034349.

BENTEN, M. S. T.; SAIT, S. M. Genetic scheduling of task graphs. *International Journal of electronics*, v. 77, n. 4, p. 401–415, 1994. Disponível em: <https://doi.org/10.1080/00207219408926072>.

BIOLCHINI, J.; MIAN, P. G.; NATALI, A. C. C.; TRAVASSOS, G. H. Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, v. 679, n. 05, p. 45, 2005.

BONYADI, M. R.; MOGHADDAM, M. E. A bipartite genetic algorithm for multi-processor task scheduling. *International Journal of Parallel Programming*, v. 37, n. 5, p. 462–487, 2009. Disponível em: <https://doi.org/10.1007/s10766-009-0107-8>.

BUDGEN, D.; BRERETON, P. Performing systematic literature reviews in software engineering. In: ACM. *Proceedings of the 28th international conference on Software engineering*. [S.l.], 2006. p. 1051–1052. Disponível em: <https://doi.org/10.1145/1134285.1134500>.

CHITRA, P.; REVATHI, S.; VENKATESH, P.; RAJARAM, R. Evolutionary algorithmic approaches for solving three objectives task scheduling problem on heterogeneous systems. In: *2nd International Advance Computing Conference*. Patiala: IEEE, 2010. p. 38–43. Disponível em: <https://doi.org/10.1109/IADCC.2010.5423042>.

CHITRA, P.; VENKATESH, P.; RAJARAM, R. Comparison of evolutionary computation algorithms for solving bi-objective task scheduling problem on heterogeneous distributed computing systems. *Sadhana*, v. 36, n. 2, p. 167–180, 2011. Disponível em: <https://doi.org/10.1007/s12046-011-0014-8>.

COELLO, C. A. C.; LAMONT, G. B.; VELDHUIZEN, D. A. V. et al. *Evolutionary algorithms for solving multi-objective problems*. [S.l.]: Springer, 2007. v. 5. Disponível em: <https://doi.org/10.1007/978-0-387-36797-2>.

CORMACK, G. V. Email spam filtering: A systematic review. *Foundations and Trends in Information Retrieval*, v. 1, n. 4, p. 335–455, 2006. Disponível em: <https://doi.org/10.1561/1500000006>.

CORRÊA, R. C.; FERREIRA, A.; REBREYEND, P. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed systems*, v. 10, n. 8, p. 825–837, 1999. Disponível em: <https://doi.org/10.1109/71.790600>.

DAOUD, M. I.; KHARMA, N. A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. *Journal of Parallel and Distributed Computing*, v. 71, n. 11, p. 1518–1531, 2011. Disponível em: <https://doi.org/10.1016/j.jpdc.2011.05.005>.

DEB, K. *Multi-Objective Optimization using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001. (Wiley Interscience Series in Systems and Optimization).

DEB, K.; AGRAWAL, R. B. Simulated binary crossover for continuous search space. *Complex Systems*, v. 9, n. 3, p. 115–148, 1994.

DEMIROZ, B.; TOPCUOGLU, H. R. Static task scheduling with a unified objective on time and resource domains. *The Computer Journal*, v. 49, n. 6, p. 731–743, 2006. Disponível em: <https://doi.org/10.1093/comjnl/bxl030>.

DHINGRA, S.; GUPTA, S. B.; BISWAS, R. Genetic algorithm parameters optimization for bi-criteria multiprocessor task scheduling using design of experiments. *International Journal of Computer, Control, Quantum and Information Engineering*, v. 8, n. 4, p. 661–667, 2014. Disponível em: <https://doi.org/10.5281/zenodo.2813952>.

EIBEN, A. E.; SMITH, J. E. et al. *Introduction to evolutionary computing*. [S.l.]: Springer Berlin Heidelberg, 2003. 300 p. (Natural Computing Series). Disponível em: <https://doi.org/10.1007/978-3-662-05094-1>. ISBN 9783662050941.

GABRIEL, P. H. R.; DELBEM, A. C. B. *Fundamentos de algoritmos evolutivos*. [S.l.]: ICMC-USP, 2008.

GANDHI, T.; NITIN; ALAM, T. Quantum genetic algorithm with rotation angle refinement for dependent task scheduling on distributed systems. In: *2017 Tenth International Conference on Contemporary Computing (IC3)*. Noida: [s.n.], 2017. p. 1–5. Disponível em: <https://doi.org/10.1109/IC3.2017.8284295>.

GARG, S. K.; BUYYA, R. Green cloud computing and environmental sustainability. *Harnessing Green IT: Principles and Practices*, Wiley Online Library, v. 2012, p. 315–340, 2012. Disponível em: <https://doi.org/10.1002/9781118305393.ch16>.

GOLDBERG, D. E.; DEB, K. A comparative analysis of selection schemes used in genetic algorithms. In: *Foundations of genetic algorithms*. [S.l.]: Elsevier, 1991. v. 1, p. 69–93. Disponível em: <https://doi.org/10.1016/B978-0-08-050684-5.50008-2>.

GOLUB, M.; KASAPOVIC, S. Scheduling multiprocessor tasks with genetic algorithms. In: HAMZA, M. H. (Ed.). *Applied Informatics*. Innsbruck: ACTA Press, 2002. p. 273–278.

GREFENSTETTE, J. Rank-based selection. *Evolutionary computation*, v. 1, p. 187–194, 2000. Disponível em: <https://doi.org/10.21236/ADA398950>.

GROUP, E.-B. M. W. et al. Evidence-based medicine. a new approach to teaching the practice of medicine. *Jama*, v. 268, n. 17, p. 2420, 1992. Disponível em: <https://doi.org/10.1001/jama.1992.03490170092032>.

GUPTA, S.; KUMAR, V.; AGARWAL, G. Task scheduling in multiprocessor system using genetic algorithm. In: *Second International Conference on Machine Learning and Computing*. Bangalore: IEEE, 2010. p. 267–271. Disponível em: <https://doi.org/10.1109/ICMLC.2010.50>.

GUZEK, M.; PECERO, J. E.; DORRONSORO, B.; BOUVRY, P. Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. *Applied Soft Computing*, v. 24, p. 432–446, 2014. Disponível em: <https://doi.org/10.1016/j.asoc.2014.07.010>.

HASSAN, M.-A.; KACEM, I.; MARTIN, S.; OSMAN, I. M. Genetic algorithms for job scheduling in cloud computing. *Studies in Informatics and Control*, v. 24, n. 4, p. 387–400, 2015. Disponível em: <https://doi.org/10.24846/v24i4y201503>.

HOLLAND, J. H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Oxford, England: U Michigan Press, 1975.

HOU, E. S. H.; ANSARI, N.; REN, H. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on parallel and distributed systems*, v. 5, n. 2, p. 113–120, 1994. Disponível em: <https://doi.org/10.1109/71.265940>.

HOU, E. S. H.; HONG, R.; ANSARI, N. Efficient multiprocessor scheduling based on genetic algorithms. In: *16th Annual Conference of IEEE Industrial Electronics Society*. Pacific Grove: IEEE, 1990. p. 1239–1243. Disponível em: <https://doi.org/10.1109/IECON.1990.149314>.

HWANG, R.; GEN, M.; KATAYAMA, H. A comparison of multiprocessor task scheduling algorithms with communication costs. *Computers & Operations Research*, v. 35, n. 3, p. 976–993, 2008. Disponível em: <https://doi.org/10.1016/j.cor.2006.05.013>.

ISERN, D.; MORENO, A. A systematic literature review of agents applied in healthcare. *J. Medical Systems*, v. 40, n. 2, p. 43:1–43:14, 2016. Disponível em: <https://doi.org/10.1007/s10916-015-0376-2>.

ISLAM, M. N. A systematic literature review of semiotics perception in user interfaces. *J. Systems and IT*, v. 15, n. 1, p. 45–77, 2013. Disponível em: <https://doi.org/10.1108/13287261311322585>.

JAIN, A.; MISHRA, M.; PEDDOJU, S. K.; JAIN, N. Energy efficient computing-green cloud computing. In: IEEE. *2013 International Conference on Energy Efficient Technologies for Sustainability*. [S.l.], 2013. p. 978–982. Disponível em: <https://doi.org/10.1109/ICEETS.2013.6533519>.

JELODAR, M. S.; FAKHRAIE, S. N.; MONTAZERI, F.; FAKHRAIE, S. M.; AHMADABADI, M. N. A representation for genetic-algorithm-based multiprocessor task scheduling. In: *IEEE Congress on Evolutionary Computation*. Vancouver: IEEE, 2006. p. 340–347. Disponível em: <https://doi.org/10.1109/CEC.2006.1688328>.

JEZIC, G.; KOSTELAC, R.; LOVREK, I.; SINKOVIC, V. Scheduling tasks with non-negligible intertask communication onto multiprocessors by using genetic algorithms. In: *Artificial Neural Nets and Genetic Algorithms*. Viena: Springer, 1999. p. 196–201. Disponível em: <https://doi.org/10.1007/978-3-7091-6384-9_34>.

KANEWALA, U.; BIEMAN, J. M. Testing scientific software: A systematic literature review. *Information & Software Technology*, v. 56, n. 10, p. 1219–1232, 2014. Disponível em: <https://doi.org/10.1016/j.infsof.2014.05.006>.

KANG, C. C.; CHUANG, Y. J.; TUNG, K. C.; CHAO, C. C.; TANG, C. Y.; PENG, S. C.; WONG, D. S. H. A genetic algorithm-based boolean delay model of intracellular signal transduction in inflammation. *BMC bioinformatics*, BioMed Central, v. 12, n. 1, p. S17, 2011. Disponível em: <https://doi.org/10.1186/1471-2105-12-S1-S17>.

KANG, Y.; ZHANG, Z.; CHEN, P. An activity-based genetic algorithm approach to multiprocessor scheduling. In: *Seventh International Conference on Natural Computation*. Berlin, Heidelberg: IEEE, 2011. v. 2, p. 1048–1052. Disponível em: <https://doi.org/10.1109/ICNC.2011.6022236>.

KAUR, K.; CHHABRA, A.; SINGH, G. Heuristics based genetic algorithm for scheduling static tasks in homogeneous parallel system. *International Journal of Computer Science and Security*, v. 4, n. 2, p. 183–198, 2010.

_____. Modified genetic algorithm for task scheduling in homogeneous parallel system using heuristics. *International Journal of Soft Computing*, v. 5, n. 2, p. 42–51, 2010. Disponível em: <https://doi.org/10.3923/ijscomp.2010.42.51>.

KAUR, R.; SINGH, G. Genetic algorithm solution for scheduling jobs in multiprocessor environment. In: *Annual IEEE India Conference*. Kochi: IEEE, 2012. p. 968–973. Disponível em: <https://doi.org/10.1109/INDCON.2012.6420757>.

KHAN, K. S.; RIET, G. T.; GLANVILLE, J.; SOWDEN, A. J.; KLEIJNEN, J. et al. *Undertaking systematic reviews of research on effectiveness: CRD's guidance for carrying out or commissioning reviews*. [S.l.]: NHS Centre for Reviews and Dissemination, 2001.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

KITCHENHAM, B.; BRERETON, O. P.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINKMAN, S. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, Elsevier, v. 51, n. 1, p. 7–15, 2009. Disponível em: <https://doi.org/10.1016/j.infsof.2008.09.009>.

KITCHENHAM, B.; BRERETON, P. A systematic review of systematic review process research in software engineering. *Information and software technology*, Elsevier, v. 55, n. 12, p. 2049–2075, 2013. Disponível em: <https://doi.org/10.1016/j.infsof.2013.07.010>.

KWOK, Y.-K.; AHMAD, I. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, v. 47, n. 1, p. 58–77, 1997. Disponível em: <https://doi.org/10.1006/jpdc.1997.1395>.

LABORATORY, K. *Standard Task Graph Set*. 2001. Acesso em: 03 abr.2019. Disponível em: <http://www.kasahara.elec.waseda.ac.jp/schedule/>.

LEE, Y.-H.; CHEN, C. A modified genetic algorithm for task scheduling in multiprocessor systems. In: *The 9th Workshop on Compiler Techniques for High-performance Computing*. [S.l.: s.n.], 2003.

LEWIS, G. A.; LAGO, P. Architectural tactics for cyber-foraging: Results of a systematic literature review. *Journal of Systems and Software*, v. 107, p. 158–186, 2015. Disponível em: <https://doi.org/10.1016/j.jss.2015.06.005>.

LIU, D.; LI, Y.; YU, M. A genetic algorithm for task scheduling in network computing environment. In: *Proceedings Fifth International Conference on Algorithms and Architectures for Parallel Processing*. Beijing: IEEE, 2002. p. 126–129. Disponível em: <https://doi.org/10.1109/ICAPP.2002.1173562>.

MALHOTRA, R. A systematic review of machine learning techniques for software fault prediction. *Appl. Soft Comput*, v. 27, p. 504–518, 2015. Disponível em: <https://doi.org/10.1016/j.asoc.2014.11.023>.

MOHAMED, M. R.; AWADALLA, M. H. A. Hybrid algorithm for multiprocessor task scheduling. *International Journal of Computer Science Issues*, v. 8, n. 3, p. 79–89, 2011.

MORADY, R.; DAL, D. A multi-population based parallel genetic algorithm for multiprocessor task scheduling with communication costs. In: CHATZIMISIOS, P.; SOYATA, T. (Ed.). *IEEE Symposium on Computers and Communication*. Messina: IEEE, 2016. p. 766–772. Disponível em: <https://doi.org/10.1109/ISCC.2016.7543829>.

OMARA, F. A.; ARAFA, M. M. Genetic algorithms for task scheduling problem. In: ABRAHAM, A.; HASSANIEN, A.-E.; SIARRY, P.; ENGELBRECHT, A. (Ed.). *Foundations of Computational Intelligence, Volume 3*. [S.l.]: Springer, 2009, (Studies in Computational Intelligence, 203). p. 479–507. Disponível em: <https://doi.org/10.1007/978-3-642-01085-9_16>.

PACHECO, M. A. C. et al. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, p. 28, 1999.

PANWAR, P.; LAL, A. K.; SINGH, J. A genetic algorithm based technique for efficient scheduling of tasks on multiprocessor system. In: *Proceedings of the International Conference on Soft Computing for Problem Solving*. New Delhi: Springer, 2012. (Advances in Intelligent and Soft Computing, 131), p. 911–919. Disponível em: <https://doi.org/10.1007/978-81-322-0491-6_84>.

PILLAI, A. S.; SINGH, K.; SARAVANAN, V.; ANPALAGAN, A.; WOUNGANG, I.; BAROLLI, L. A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems. *Soft Computing*, v. 22, n. 10, p. 3271–3285, 2018. Disponível em: <https://doi.org/10.1007/s00500-017-2789-y>.

POP, F.; DOBRE, C.; CRISTEA, V. Genetic algorithm for DAG scheduling in grid environments. In: LETIA, I. A. (Ed.). *Proceedings of the International Conference on Intelligent Computer Communication and Processing*. Cluj-Napoca: IEEE, 2009. p. 299–305. Disponível em: <https://doi.org/10.1109/ICCP.2009.5284747>.

RAMACHANDRA, G.; ELMAGHRABY, S. E. Sequencing precedence-related jobs on two machines to minimize the weighted completion time. *International Journal of Production Economics*, v. 100, n. 1, p. 44–58, 2006. Disponível em: <https://doi.org/10.1016/j.ijpe.2004.10.014>.

RAZALI, N. M.; GERAGHTY, J. et al. Genetic algorithm performance with different selection strategies in solving tsp. In: INTERNATIONAL ASSOCIATION OF ENGINEERS HONG KONG. *Proceedings of the world congress on engineering*. [S.l.], 2011. v. 2, n. 1, p. 1–6.

RUSSELL, S.; NORVIG, P.; DAVIS, E. *Artificial Intelligence: A Modern Approach*. [S.l.]: Prentice Hall, 2010. 1132 p. (Prentice Hall series in artificial intelligence). ISBN 9780136042594.

SAHIN, F.; YAN, Z. Mobile phones in data collection: A systematic review. *IJCBPL*, v. 3, n. 3, p. 67–87, 2013. Disponível em: <https://doi.org/10.4018/ijcbpl.2013070106>.

SATHAPPAN, O. L.; CHITRA, P.; VENKATESH, P.; PRABHU, M. Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system. *International Journal of Information Technology, Communications and Convergence*, v. 1, n. 2, p. 146–158, 2011. Disponível em: <https://doi.org/10.1504/IJITCC.2011.039282>.

SHEIKH, H. F.; AHMAD, I.; ARSHAD, S. A. Performance, energy, and temperature enabled task scheduling using evolutionary techniques. *Sustainable Computing: Informatics and Systems*, v. 22, p. 272–286, jun. 2017. Disponível em: <https://doi.org/10.1016/j.suscom.2017.10.002>.

SHEIKH, H. F.; AHMAD, I.; FAN, D. An evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors. *IEEE Transactions on Parallel and Distributed Systems*, v. 27, n. 3, p. 668–681, 2016. Disponível em: <https://doi.org/10.1109/TPDS.2015.2421352>.

SILVA, E. C. da; GABRIEL, P. H. R. Genetic algorithms and multiprocessor task scheduling: A systematic literature review. In: *The 16th National Meeting on Artificial and Computational Intelligence*. Salvador, BA, Brazil: Sociedade Brasileira de Computação, 2019. v. 1, n. 1, p. 1–12.

SINGH, H.; YOUSSEF, A. Mapping and scheduling heterogeneous task graphs using genetic algorithms. In: *Proceedings of the Tenth International Parallel Processing Symposium*. Honolulu: IEEE, 1996. p. 86–97.

SINGH, J.; SINGH, G. Improved task scheduling on parallel system using genetic algorithm. *International Journal of Computer Applications*, v. 39, n. 17, p. 17–22, 2012. Disponível em: <https://doi.org/10.5120/4912-7449>.

SINGH, K.; PILLAI, A. S. Schedule length optimization by elite-genetic algorithm using rank based selection for multiprocessor systems. In: *International Conference on Embedded Systems*. Coimbatore: IEEE, 2014. p. 86–91. Disponível em: <https://doi.org/10.1109/EmbeddedSys.2014.6953096>.

SINGH, R. Task scheduling with genetic approach and task duplication technique. *International Journal of Computer Applications & Information Technology*, v. 1, n. 1, p. 7–12, 2012.

TOPCUOGLU, H.; SEVILMIS, C. Task scheduling with conflicting objectives. In: YAKHNO, T. (Ed.). *Advances in Information Systems*. Berlin, Heidelberg: Springer, 2002. (Lecture Notes in Computer Science, 2457), p. 346–355. Disponível em: <https://doi.org/10.1007/3-540-36077-8_36>.

TSAFNAT, G.; GLASZIOU, P.; CHOONG, M.; DUNN, A.; GALGANI, F.; COIERA, E. Systematic review automation technologies. BioMed Central Ltd., 2014. Disponível em: <https://doi.org/10.1186/2046-4053-3-74>.

TSUCHIYA, T.; OSADA, T.; KIKUNO, T. A new heuristic algorithm based on GAs for multiprocessor scheduling with task duplication. In: GOSCINSLCI, A.; HOBBS, M.; ZHOU, W. (Ed.). *Proceedings of 3rd International Conference on Algorithms and Architectures for Parallel Processing*. Melbourne: IEEE, 1997. p. 295–308.

_____. Genetics-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems*, v. 22, n. 3-4, p. 197–207, 1998. Disponível em: <https://doi.org/10.1016/S0141-9331(98)00079-9>.

TSUJIMURA, Y.; GEN, M. Genetic algorithms for solving multiprocessor scheduling problems. In: YAO, X.; KIM, J.-H.; FURUHASHI, T. (Ed.). *Simulated Evolution and Learning*. Berlin, Heidelberg: Springer, 1996. (Lecture Notes in Computer Science, 1285), p. 106–115. Disponível em: <https://doi.org/10.1007/BFb0028527>.

WALL, M. B. *A genetic algorithm for resource-constrained scheduling*. Tese (Doutorado) — Massachusetts Institute of Technology, 1996.

WANG, L.; SIEGEL, H. J.; ROYCHOWDHURY, V. P. A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments. In: *Proceedings of the Heterogeneous Computing Workshop*. [S.l.: s.n.], 1996. p. 72–85.

WANG, L.; SIEGEL, H. J.; ROYCHOWDHURY, V. P.; MACIEJEWSKI, A. A. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of parallel and distributed computing*, v. 47, n. 1, p. 8–22, nov. 1997. Disponível em: <https://doi.org/10.1006/jpdc.1997.1392>.

WANG, P.-C.; KORFHAGE, W. Process scheduling using genetic algorithms. In: *Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing*. San Antonio: IEEE, 1995. p. 638–641. Disponível em: <https://doi.org/10.1109/SPDP.1995.530742>.

WANG, X.; YEO, C. S.; BUYYA, R.; SU, J. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Generation Computer Systems*, v. 27, n. 8, p. 1124–1134, out. 2011. Disponível em: <https://doi.org/10.1016/j.future.2011.03.008>.

WOO, S.-H.; YANG, S.-B.; KIM, S.-D.; HAN, T.-D. Task scheduling in distributed computing systems with a genetic algorithm. In: *Proceedings of the High Performance Computing on the Information Superhighway*. Seoul: IEEE, 1997. p. 301–305. Disponível em: <https://doi.org/10.1109/HPC.1997.592164>.

WOODALL, P.; BRERETON, P. A systematic literature review of inference strategies. *International Journal of Information and Computer Security*, v. 4, n. 2, 2010. Disponível em: <https://doi.org/10.1504/IJICS.2010.034813>.

WU, A. S.; YU, H.; JIN, S.; LIN, K.-C.; SCHIAVONE, G. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on parallel and distributed systems*, v. 15, n. 9, p. 824–834, set. 2004. Disponível em: <https://doi.org/10.1109/TPDS.2004.38>.

XU, Y.; LI, K.; HU, J.; LI, K. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, v. 270, p. 255–287, jun. 2014. Disponível em: <https://doi.org/10.1016/j.ins.2014.02.122>.

XU, Y.; LI, K.; KHAC, T. T.; QIU, M. A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems. In: *International Conference on High Performance Computing and Communication & International Conference on Embedded Software and Systems*. Liverpool: IEEE, 2012. p. 639–646. Disponível em: <https://doi.org/10.1109/HPCC.2012.91>.

YAO, W.; YOU, J.; LI, B. Main sequences genetic scheduling for multiprocessor systems using task duplication. *Microprocessors and Microsystems*, v. 28, n. 2, p. 85–94, 2004. Disponível em: <https://doi.org/10.1016/j.micpro.2003.10.002>.

ZHONG, Y.-W.; YANG, J.-G. A genetic algorithm for tasks scheduling in parallel multiprocessor systems. In: *Proceedings of the International Conference on Machine Learning and Cybernetics*. XI'an: IEEE, 2003. v. 3, p. 1785–1790. Disponível em: <https://doi.org/10.1109/ICMLC.2003.1259786>.

ZHU, C.; LEUNG, V. C.; SHU, L.; NGAI, E. C.-H. Green internet of things for smart world. *IEEE Access*, IEEE, v. 3, p. 2151–2162, 2015. Disponível em: <https://doi.org/10.1109/ACCESS.2015.2497312>.

ZOMAYA, A. Y.; WARD, C.; MACEY, B. Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel & Distributed Systems*, v. 10, n. 8, p. 795–812, ago. 1999. Disponível em: <https://doi.org/10.1109/71.790598>.