UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOÃO PEDRO GONÇALVES MOREIRA

Balanceamento de linhas de produção usando regras de seleção descobertas por Genetic Programming

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Marcus Ritt

Porto Alegre
2020

RESUMO

Linhas de montagem são uma forma de produção em massa na qual um produto passa por uma sequência de estações onde são realizadas tarefas necessárias à sua confecção. É comum o uso de linhas de montagem em centros de trabalho para pessoas com deficiência (CTDs), pois a divisão da produção em pequenas tarefas facilita a atribuição de tarefas a trabalhadores de acordo com suas características. O *Problema de balanceamento de linhas de produção e designação de trabalhadores* (ALWABP) foi criado a partir da necessidade de otimizar a produção de linhas de montagem em CTDs, ao mesmo tempo em que a atribuição de tarefas a trabalhadores nestas linhas seja feita considerando características específicas de cada trabalhador. Para obter soluções de alta qualidade para instâncias do ALWABP, é comum o uso de métodos heurísticos baseados em regras de prioridade. Ao analisar regras de prioridade existentes na literatura do ALWABP, percebe-se um conjunto de elementos comuns que combinados de diferentes formas dão origem às diversas regras já existentes. Ao criar uma linguagem capaz de descrever e combinar estes elementos comuns de diferentes formas, é possível representar regras de prioridade como programas de computador, o que permite a aplicação da técnica de computação evolutiva denominada *Programação Genética* para evoluir programas e descobrir de forma automática novas regras de prioridade de mais alta qualidade. A aplicação de regras de prioridade descobertas por programação genética, em conjunto com algoritmos que são o estado da arte em termos de métodos heurísticos para o ALWABP, constitui uma ferramenta poderosa para encontrar de forma rápida soluções eficientes para instâncias arbitrárias do problema.

Palavras-chave: Programação genética. computação evolutiva. algoritmos genéticos. regras de prioridade. balanceamento de linhas de montagem. ALWABP.

Assembly line balancing using selection rules discovered by Genetic Programming

ABSTRACT

Assembly lines are a way of large-scale production where a product passes through a sequence of stations on which tasks necessary to its assembly are performed. It is common to use assembly lines in sheltered work centers for people with disabilities (SWDs), since the division of the production in small tasks facilitates the assignment of tasks to workers according to their specific characteristics. The *Assembly line worker assignment and balancing problem* (ALWABP) was created due to the necessity of optimizing the production of assembly lines in SWDs, considering the specific characteristics of each worker when assigning tasks to workers in such lines. To produce high-quality solutions for ALWABP instances, it is common to use heuristic methods based on priority rules. After analyzing priority rules described in the ALWABP literature, it is possible to identify a set of common elements, such that the rules observed may be described simply as different combinations of these elements. By creating a language capable of describing and combining the common elements in different ways, it is possible to represent priority rules as computer programs, which allows the application of the evolutionary computation technique called *Genetic Programming* to evolve programs and automatically discover new higher-quality priority rules. The application of priority rules discovered by genetic programming, together with algorithms that are the state of the art in terms of heuristic methods for the ALWABP, constitutes a powerful tool to quickly find efficient solutions to arbitrary instances of the problem.

Keywords: Genetic programming. evolutionary computation. genetic algorithms. priority rules. assembly line balancing. ALWABP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CTD	Centro de Trabalho para pessoas com Deficiência
SWD	Sheltered Work center for Disabled
ALWABP	Assembly Line Worker Assignment and Balancing Problem
SALBP	Simple Assembly Line Balancing Problem
SoAP	Station-oriented Assignment Procedure
IPBS	Iterated Probabilistic Beam Search

SUMÁRIO

1 INTRODUÇÃO

Países que apresentam altos índices de desenvolvimento como Japão, Inglaterra e Espanha (ARAÚJO, 2011), promovem a manutenção de centros de trabalho específicos para pessoas com deficiência (CTDs), como estratégia para a integração deste público ao mercado de trabalho. Em CTDs é comum o uso de linhas de montagem, pois a divisão da produção característica deste modelo facilita a atribuição de tarefas de acordo com especificidades de cada trabalhador (CRISTÓBAL et al., 2007).

O *Problema de balanceamento de linhas de produção e designação de trabalhadores* (ALWABP, do inglês *Assembly Line Worker Assignment and Balancing Problem*) foi proposto por CRISTÓBAL et al. (2007) durante um estudo de caso em uma linha de montagem em um CTD na Espanha. No ALWABP, o objetivo é encontrar uma atribuição de tarefas a trabalhadores de uma linha de montagem, de forma a maximizar a produtividade da linha, respeitando restrições relacionadas a incompatibilidades entre tarefas e trabalhadores e relações de precedência entre tarefas.

ALWABP é uma generalização de um outro problema NP-difícil relacionado a linhas de montagem, o *Problema simples de balanceamento de linhas de produção* (SALBP, *Simple Assembly Line Balancing Problem*) (SALVESON, 1955). Por este motivo, grande parte dos estudos sobre o ALWABP visam o desenvolvimento de métodos heurísticos capazes de encontrar soluções de alta qualidade em tempo razoável para instâncias arbitrárias. Os algoritmos mais bem sucedidos aplicados ao ALWABP são *métodos baseados em regras de prioridade* (OTTO; OTTO, 2013), sendo que o uso de diferentes regras de prioridade por estes métodos afeta significativamente a qualidade das soluções produzidas (MOREIRA et al., 2012).

Ao analisar regras de prioridade presentes na literatura do ALWABP, é possível identificar elementos comuns que podem ser recombinados para a produção de novas regras. A elaboração de uma linguagem capaz de descrever regras de prioridade como combinações destes elementos permite a aplicação da técnica de computação evolutiva denominada *Programação Genética* (ou *Genetic Programming*, em inglês) para a obtenção automática de novas regras que, ao serem aplicadas em métodos heurísticos, têm o potencial de elevar a qualidade das soluções produzidas por estes métodos.

1.1 Objetivos

- Propor uma linguagem para a representação de regras de seleção de tarefas no contexto do ALWABP.
- Mostrar que as regras descobertas por programação genética são competitivas com as melhores regras de seleção desenvolvidas manualmente já descritas na literatura.
- Comparar a qualidade de soluções obtidas por métodos heurísticos que são o estado da arte para o ALWABP (BLUM; MIRALLES, 2011) (BORBA; RITT, 2014), utilizando regras descobertas e regras já existentes.

1.2 Estrutura do texto

No Capítulo 2 são apresentados o ALWABP e o problema simples de balanceamento de linhas de produção (SALBP), assim como as principais variações destes problemas, e conceitos fundamentais sobre linhas de montagem. O Capítulo 3 aborda um procedimento baseado em regras de prioridade utilizado pelos métodos heurísticos mais eficientes para o ALWABP.

No Capítulo 4 são apresentadas as regras de seleção de tarefas, objeto de estudo deste trabalho, assim como regras clássicas da literatura e elementos comuns que compõem estas regras. O Capítulo 5 mostra a linguagem proposta para representar funções de prioridade utilizadas por regras de seleção de tarefas. Capítulos 6 e 7 apresentam conceitos básicos sobre algoritmos genéticos e programação genética, e descrevem os elementos do algoritmo genético implementado neste trabalho para evoluir funções de prioridade.

O Capítulo 8 descreve uma série de estratégias lógicas que podem ser aplicadas para ajudar métodos heurísticos a encontrar melhores soluções para o ALWABP. No Capítulo 9 são apresentados resultados de experimentos computacionais, assim como a melhor regra encontrada durante os experimentos. O Capítulo 10 apresenta as conclusões do trabalho.

2 BALANCEAMENTO DE LINHAS DE PRODUÇÃO

Este capítulo descreve o problema simples de balanceamento de linhas de produção (SALBP), o problema de balanceamento de linhas de produção e designação de trabalhadores (ALWABP), assim como as principais variações definidas para estes problemas. Será explicado o modelo de linhas de produção utilizado pelo SALBP e pelo ALWABP.

Será descrita a variação do ALWABP mais trabalhada na literatura, o ALWABP-2, e o motivo de haver maior interesse por esta variação. Ao final, será mostrado que a variante ALWABP-2 é uma generalização da variante SALBP-2.

2.1 Problema simples de balanceamento de linhas de produção

Uma instância do SALBP contém um conjunto T das tarefas de uma linha de produção, e para cada tarefa $i \in T$ existe um número inteiro t_i que indica o tempo necessário para a sua execução. Relações de precedência entre tarefas são descritas por um grafo direcionado acíclico $D = (T, A)$, de tal forma que um arco $(i, j) \in A$ indica que a tarefa j é sucessora imediata da tarefa i, portanto i deve ser concluída antes que j possa começar a ser executada.

Para descrever relações transitivas de precedência, é útil definir um conjunto auxiliar A^* de pares de tarefas, de forma que um par $(i, j) \in A^*$ indica que existe um caminho em D que parte da tarefa i e chega na tarefa j. A seguinte notação será utilizada neste trabalho para relações de precedência envolvendo uma tarefa qualquer $i \in T$:

$F_i = \{ j \mid (i,j) \in A \}$	Conjunto dos sucessores imediatos.
$F_i^* = \{ j \mid (i,j) \in A^* \}$	Conjunto de todos os sucessores.
$P_i = \{ j \mid (j,i) \in A \}$	Conjunto dos predecessores imediatos.
$P_i^* = \{ j \mid (j,i) \in A^* \}$	Conjunto de todos os predecessores.

Uma solução para uma instância do SALBP é uma linha de montagem composta por um conjunto S de estações de trabalho e um mapeamento que associa cada uma das tarefas em T a uma única estação em S. O *workload* de uma estação de trabalho é igual à soma dos tempos de execução das tarefas que estão na estação. O *tempo de ciclo* de uma linha de montagem, notado por C, é definido como o maior workload dentre as estações que compõem a linha.

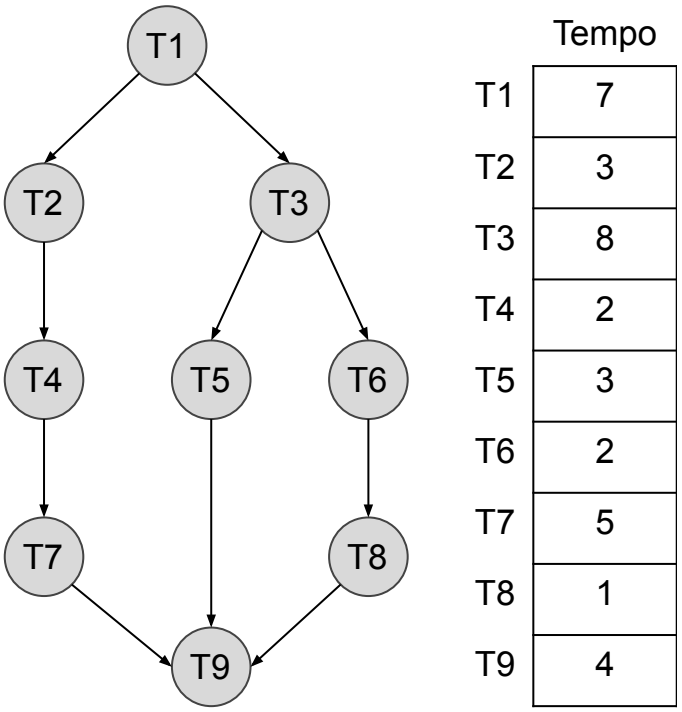


Figura 2.1 – Exemplo de instância do SALBP, mostrando o grafo de precedências e o tempo de execução de cada tarefa.
Fonte: O Autor.

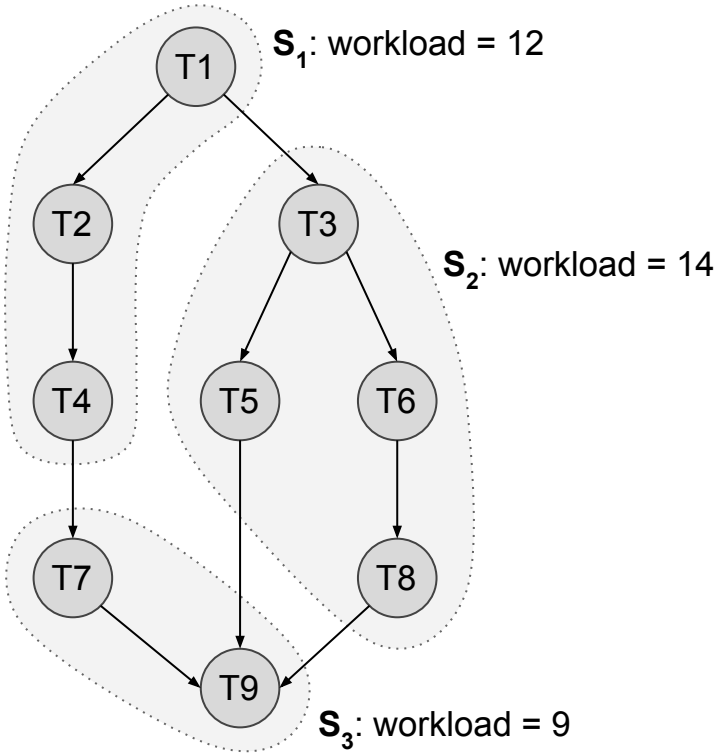


Figura 2.2 – Uma solução para a instância do SALBP com estações S_1, S_2 e S_3, e tempo de ciclo $C = 14$.
Fonte: O Autor.

As relações de precedência entre tarefas produzem também uma relação de ordem parcial entre as estações S de uma solução. A divisão de tarefas em estações de trabalho deve respeitar as precedências definidas em D. Isso implica que se uma tarefa $i \in T$ é alocada para uma estação $s \in S$, então as tarefas em P_i^* podem ser alocadas apenas para o conjunto de estações que contém s e suas estações predecessoras. A Figura 2.1 ilustra uma instância do SALBP e uma possível solução com tempo de ciclo $C = 14$ é apresentada na Figura 2.2.

2.2 Modelo de linhas de montagem

No modelo de linhas de montagem usado no SALBP e no ALWABP, cada estação de trabalho contém um único trabalhador, que realiza sequencialmente todas as tarefas alocadas na estação. Trabalhadores em estações diferentes podem executar tarefas de forma simultânea.

O funcionamento de uma linha de montagem ocorre em forma de um pipeline, onde cada estação recebe um produto parcialmente completo vindo do estágio anterior, realiza uma sequência de operações sobre o produto, e passa o resultado para o estágio seguinte, até que o produto finalizado saia da última estação da linha. O tempo necessário para transportar um produto de uma estação a outra é desconsiderado.

Este funcionamento em pipeline garante uma propriedade importante: o tempo entre duas saídas consecutivas de produtos da última estação vai ser igual ao maior workload dentre as estações que compõem a linha, ou seja, será igual ao tempo de ciclo C. Maximizar a taxa de produção de uma linha de montagem é, portanto, equivalente a minimizar o seu tempo de ciclo. Balancear uma linha de montagem significa dividir suas tarefas em estações de trabalho, de forma a obter o menor tempo de ciclo possível para a linha.

2.3 Variações do SALBP

Além das restrições apresentadas anteriormente sobre precedências de tarefas, a definição de solução viável ou ótima para o SALBP ainda depende da variante considerada. Existem quatro variantes principais: o SALBP-1, SALBP-2, SALBP-E e SALBP-F.

No SALBP-1, o objetivo é minimizar o número de estações de trabalho na solução,

dado um limite superior C para o tempo de ciclo. Visto que cada estação contém um único trabalhador, esta variante é equivalente a encontrar uma solução que minimize o número de trabalhadores necessários na linha de montagem, sem que alguma estação tenha workload maior que C. No SALBP-2, cada instância possui um número fixo de estações de trabalho, e o objetivo é encontrar uma solução que minimiza o tempo de ciclo da linha. SALBP-E tem por objetivo minimizar o valor do produto $|S|C$, e SALBP-F é um problema de decisão que consiste em determinar se existe alguma solução para determinados valores de $|S|$ e C.

O SALBP-1 é uma generalização de um outro problema NP-difícil conhecido, o *problema do empacotamento* (bin packing). No problema do empacotamento existe um conjunto de itens (tarefas), cada item possuindo um volume (tempo de execução), que devem ser empacotados em um conjunto de recipientes (estações) de mesmo volume (limite no tempo de ciclo), com o objetivo de utilizar o menor número de recipientes possível. Uma instância do SALBP-1 que não tenha relações de precedência satisfaz esta definição. Outras variações do SALBP podem ser vistas como casos particulares do *problema da partição*, que consiste em particionar um conjunto de elementos em subconjuntos disjuntos de forma a otimizar determinado critério. Baybars (1986) apresenta em detalhes a relação das diferentes variações do SALBP com outros problemas NP-difíceis conhecidos.

2.4 Problema de balanceamento de linhas de produção e designação de trabalhadores

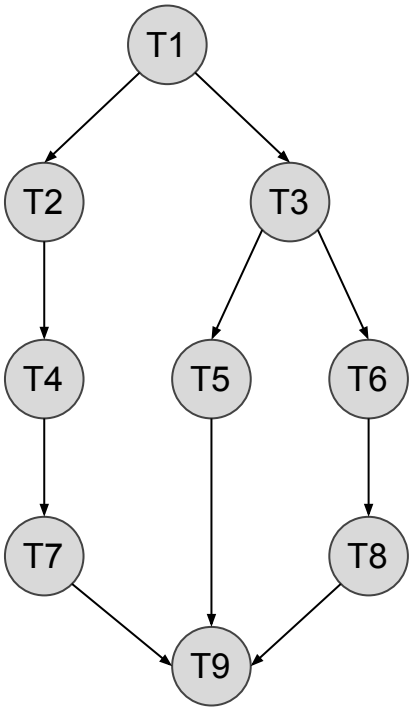
Assim como no SALBP, uma instância do ALWABP contém um conjunto T de tarefas de uma linha de montagem, e as relações de precedência entre estas tarefas são também descritas por um grafo direcionado acíclico $D = (T, A)$. Porém, no ALWABP existe um conjunto de trabalhadores W, e cada tarefa agora não tem mais um tempo fixo de execução: o tempo de execução de uma tarefa vai depender do trabalhador que irá realizar a tarefa. Para cada par $(i, w) \in T \times W$, o valor t_{wi} indica o tempo que o trabalhador w demora para completar a tarefa i.

A estratégia de atribuir tempos diferentes para trabalhadores ao executar tarefas foi elaborada por CRISTÓBAL et al. (2007), como forma de modelar diferentes níveis de habilidade de trabalhadores na realização de tarefas em CTDs. Existem casos onde um trabalhador w não pode ser associado a uma tarefa i específica, e estas situações são

modeladas atribuindo-se $t_{wi} = \infty$. Esta flexibilidade na modelagem de uma instância do ALWABP pode ser usada para outras finalidades, como por exemplo garantir que uma determinada tarefa será realizada por um trabalhador específico por motivos terapêuticos.

Uma solução ótima para uma instância do ALWABP-2 é uma alocação das tarefas de T a um conjunto de $|W|$ estações de trabalho, que respeite as restrições de precedência entre tarefas e minimize o tempo de ciclo da linha de montagem. Cada estação de trabalho está associada a um trabalhador, e o workload de uma estação é a soma dos tempos de suas tarefas para o trabalhador alocado à estação. A Figura 2.3 mostra uma instância do ALWABP, e a Figura 2.4 uma solução possível com tempo de ciclo $C = 26$.

Assim como no SALBP, é possível também definir variações -1, -2, -E e -F para o ALWABP. Porém, a variante ALWABP-2 é a mais interessante para a aplicação em CTDs, visto que a finalidade destes centros é integrar pessoas com deficiência ao mercado de trabalho, ao invés de tentar reduzir o número de trabalhadores na linha. Portanto, existe um conjunto conhecido de trabalhadores para os quais se quer encontrar uma solução que maximiza a produtividade, de forma que todos fiquem encarregados da realização de tarefas. Dito isto, o estudo apresentado neste trabalho dedica-se ao ALWABP-2.



	1	2	3
T1	2	11	∞
T2	4	3	7
T3	54	2	4
T4	7	∞	∞
T5	5	1	12
T6	3	9	10
T7	18	32	43
T8	∞	2	15
T9	19	3	5

Figura 2.3 – Exemplo de instância do ALWABP. O valor na linha i e coluna w é o tempo que o trabalhador w demora para completar a tarefa i.
Fonte: O Autor.

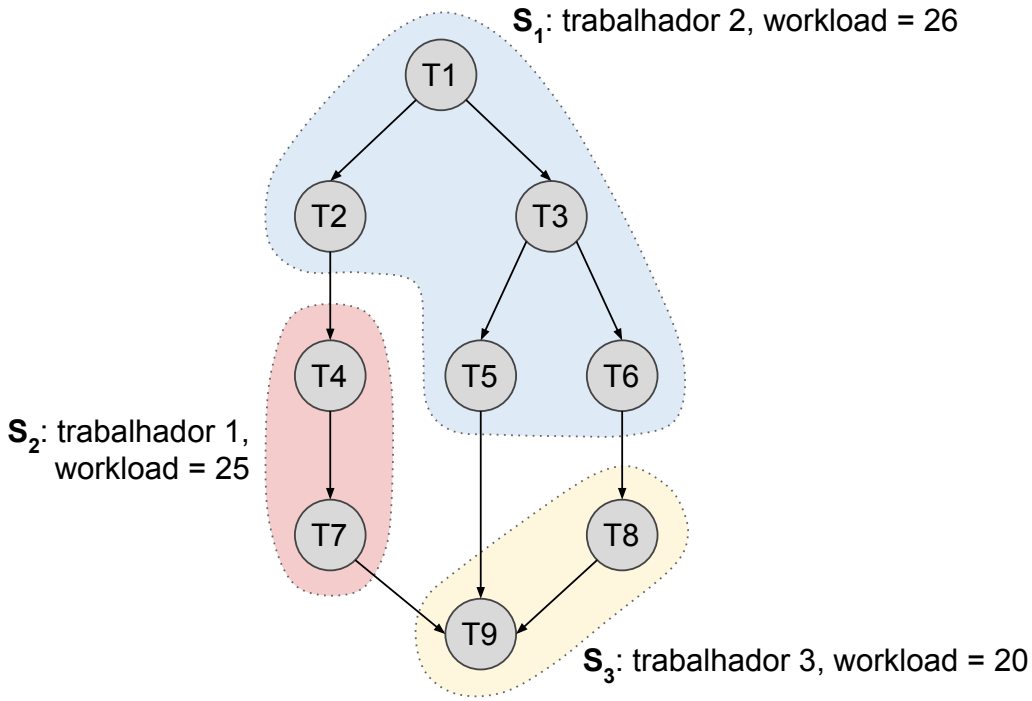


Figura 2.4 – Uma solução para a instância do ALWABP com tempo de ciclo $C = 26$.
Fonte: O Autor.

2.5 ALWABP-2 como uma generalização do SALBP-2

É possível construir uma instância do ALWABP-2 a partir de uma instância qualquer do SALBP-2, de forma que o tempo de ciclo ótimo se mantenha o mesmo. Este processo mantém o grafo de precedências D e o conjunto de tarefas T inalterados.

Defina o conjunto W como contendo um número de trabalhadores exatamente igual ao valor fixado para o número de estações na instância do SALBP-2. O tempo de execução t_i de cada tarefa $i \in T$ na instância do SALBP-2 deve ser atribuído para cada trabalhador $w \in W$ na instância do ALWABP-2, ou seja:

$$\forall i \in T, \forall w \in W : t_{wi} = t_i$$

De posse de um algoritmo que encontra o tempo de ciclo ótimo para qualquer instância do ALWABP-2, é possível também encontrar o tempo de ciclo ótimo para qualquer instância do SALBP-2 por meio da transformação apresentada. A Figura 2.5 mostra à esquerda uma instância do SALBP-2 e à direita a instância do ALWABP-2 correspondente.

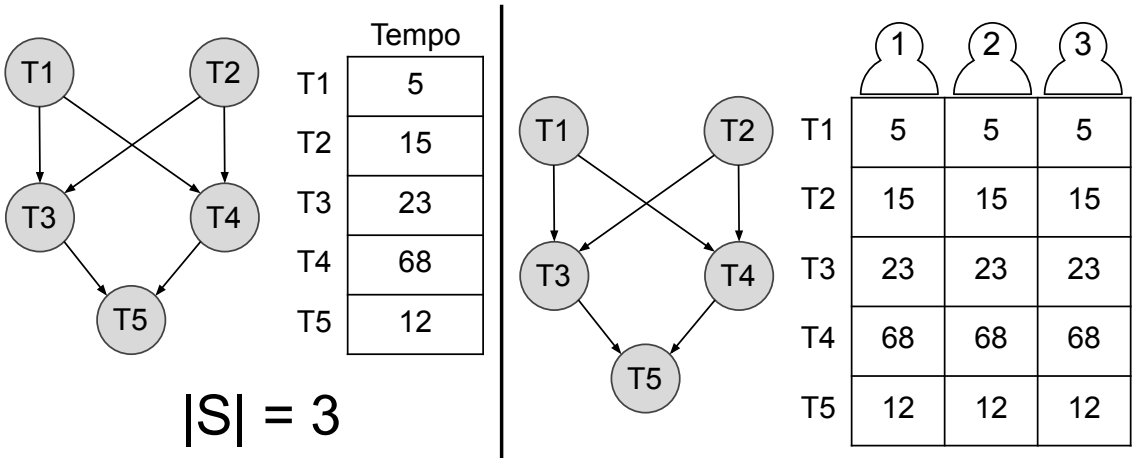


Figura 2.5 – Exemplo de instância do SALBP-2 como caso particular do ALWABP-2.
Fonte: O Autor.

3 PROCEDIMENTO DE ALOCAÇÃO ORIENTADO A ESTAÇÕES

Os métodos heurísticos mais bem sucedidos para o ALWABP-2 utilizam um *Procedimento de alocação orientado a estações* (SoAP, *Station-oriented Assignment Procedure*) (MUTLU; POLAT; AYCA, 2013) (MOREIRA et al., 2012) (POLAT et al., 2016) . Este procedimento tenta construir uma solução válida para uma instância do ALWABP-2 adicionando uma estação de trabalho à solução por vez, de forma que nenhuma estação tenha um workload maior do que um limite superior para o tempo de ciclo C. Seu resultado é ou uma solução viável encontrada, ou uma informação indicando que o procedimento não conseguiu produzir solução viável.

Para entender melhor como um SoAP funciona, é necessário apresentar alguns conceitos relacionados. Uma tarefa está *disponível* quando ela ainda não foi alocada para uma estação, porém seus predecessores, caso houver, já foram alocados. Similarmente, um trabalhador está disponível quando ele ainda não foi alocado para nenhuma estação de trabalho já adicionada à solução.

O que um SoAP faz é repetir uma sequência de passos, onde a cada passo ele constrói uma estação de trabalho para cada trabalhador disponível e escolhe uma destas estações para adicionar à solução. Para construir uma estação para um trabalhador, o SoAP repetidamente escolhe uma tarefa disponível e a adiciona à estação, até que não exista mais nenhuma tarefa disponível que possa ser adicionada sem que o workload da estação ultrapasse o limite superior no tempo de ciclo C. Para escolher a próxima tarefa a ser adicionada, o SoAP usa uma *regra de seleção de tarefas*, e para selecionar a estação a ser adicionada à solução, usa uma *regra de seleção de trabalhadores*.

Visto que o SoAP é usado apenas como uma tentativa de construir uma solução viável para um dado C, é necessário que ele seja utilizado como parte de um procedimento que faz uma busca no espaço de tempos de ciclo possíveis, tentando encontrar de alguma forma o menor tempo de ciclo para o qual o SoAP consegue produzir uma solução viável (SCHOLL; VOSS, 1996). A heurística construtiva proposta por MOREIRA et al. (2012) utiliza uma *busca por limite inferior*, onde parte-se de um valor inicial para o tempo de ciclo C, e caso o SoAP não consiga encontrar solução viável, incrementa-se este valor e repete-se o procedimento, até que uma solução viável seja encontrada.

O Algoritmo 1 mostra o pseudocódigo da heurística construtiva. Neste pseudocódigo e no que segue deste trabalho, U_w representa o conjunto de trabalhadores atualmente disponíveis durante a execução do SoAP.

Algoritmo 1: Heurística construtiva para o ALWABP-2.

Entrada: Instância do ALWABP.

Saída: Menor tempo de ciclo para o qual a heurística encontrou solução viável.

```
1  início
2      C ← limite inferior para o tempo de ciclo ótimo
3      enquanto solução não encontrada faça
4          Trabalhadores disponíveis U_w ← W
5          Tarefas não atribuídas U_t ← T
6          Estações da solução S ← ∅
7          enquanto U_w ≠ ∅ faça
8              Estações candidatas S_c ← ∅
9              para cada trabalhador w ∈ U_w faça
10                 Abra uma estação vazia K
11                 Atribua w para a estação K
12                 enquanto existe tarefa disponível i tal que t_wi + K.workload ≤ C faça
13                     Calcule a prioridade das tarefas disponíveis que cabem em K
14                     Adicione a tarefa mais prioritária à estação K
15                 S_c ← S_c ∪ {K}
16             Selecione uma estação S_s ∈ S_c usando a regra de seleção de trabalhadores
17             U_w ← U_w − {W_s}, onde W_s é o trabalhador em S_s
18             U_t ← U_t − T_s, onde T_s é o conjunto de tarefas em S_s
19             S ← S ∪ {S_s}
20         se U_t = ∅ então
21             retorna C
22         senão
23             C ← C + 1
```

4 REGRAS DE SELEÇÃO DE TAREFAS

Como discutido anteriormente, um SoAP utiliza no seu processo de decisão regras de seleção para determinar quais tarefas e trabalhadores escolher durante a construção de uma solução. É conhecido que diferentes regras influenciam nos valores de C para os quais o procedimento consegue encontrar solução viável (MOREIRA et al., 2012), (BORBA; RITT, 2014), (MOREIRA; RITT, 2019), impactando portanto a qualidade das soluções encontradas por métodos que usam um SoAP como componente.

A análise realizada neste estudo é focada sobre regras de seleção de tarefas. Nesta seção, serão apresentadas regras descritas na literatura do ALWABP, e também a identificação de características elementares que constituem regras de seleção de tarefas.

4.1 Definição

Uma *regra de seleção de tarefas* é do tipo $\max f(i)$ ou $\min f(i)$, onde $f(i)$ é uma *função de prioridade de tarefas*, calculada com base em atributos de uma tarefa $i \in T$. Existem dois tipos principais de atributos associados a tarefas: atributos estruturais e temporais.

4.2 Atributos estruturais

Atributos estruturais estão associados às relações de precedência entre tarefas e características do grafo direcionado acíclico D que descreve estas relações. Por exemplo, o número de sucessores imediatos $|F_i|$ e o número total de sucessores $|F_i^*|$ são atributos estruturais da tarefa i. Portanto, uma regra de seleção $\max |F_i|$ selecionaria como tarefa para ser adicionada à estação sendo construída aquela que, dentre as tarefas disponíveis que cabem na estação, tiver o maior número de sucessores imediatos, enquanto que $\max |F_i^*|$ selecionaria a que tem o maior número total de sucessores.

Para ilustrar, na Figura 4.1 se T_a e T_b são as tarefas disponíveis no momento que cabem na estação sendo construída sem que o workload ultrapasse C, a regra $\max |F_i|$ selecionaria a tarefa T_a para adicionar à estação, enquanto que $\max |F_i^*|$ selecionaria T_b.

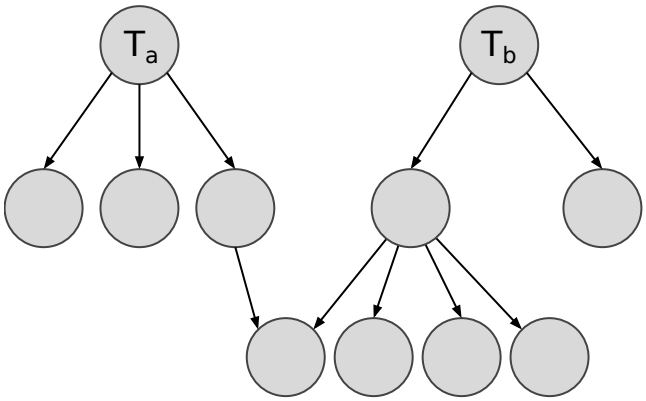


Figura 4.1 – Grafo de precedências para comparar as regras $\max |F_i|$ e $\max |F_i^*|$.
Fonte: O Autor.

4.3 Atributos temporais

Atributos temporais são valores relacionados aos tempos de execução das tarefas. Por exemplo, sendo w o trabalhador para o qual está sendo construída a estação atual durante o SoAP, t_{wi} pode ser definido como um atributo da tarefa i. Outro exemplo de atributo temporal que pode ser definido, e que de fato é usado por uma das regras de seleção mais eficientes da literatura, é o menor tempo de execução dentre os trabalhadores disponíveis, desconsiderando o tempo do trabalhador atual: $\min_{j \in U_w - \{ w \}} t_{ji}$

4.4 Regras da literatura

Várias regras presentes na literatura do ALWABP foram adaptadas de regras existentes para o SALBP, onde algumas se destacam por apresentar geralmente um desempenho superior (SCHOLL, 1999) (SCHOLL; BECKER, 2006). A Tabela 4.1 mostra algumas das regras mais importantes da literatura do ALWABP para seleção de tarefas. Na tabela, Max ou Min no nome da regra indicam se a sua função de prioridade deve ser maximizada ou minimizada. Na descrição das funções de prioridade, as notações novas t_j^- e t_j^+ indicam respectivamente o menor e o maior tempo de execução da tarefa j entre os trabalhadores disponíveis U_w, sem contar o tempo do trabalhador da estação atual w. A tarefa para a qual está sendo calculado o valor de prioridade é notada por i.

As regras MaxF e MaxIF usam atributos estruturais de tarefas. MinTime e MinRank usam atributos temporais. MaxPW$^-$ e MaxPW$^+$ usam tanto atributos estruturais quanto temporais, pois combinam somas de tempos de execução sobre conjuntos associ-

Tabela 4.1 – Regras clássicas para seleção de tarefas na literatura do ALWABP.

Regra	Função de prioridade
MaxF	F = $\lvert F_i^* \rvert$
MaxIF	IF = $\lvert F_i \rvert$
MinTime	Time = t_{wi}
MinRank	Rank = $\lvert \{ w' \in U_w \mid t_{w'i} < t_{wi} \} \rvert$
MaxPW$^-$	PW$^-$ = $\sum_{j \in F_i^* \cup \{i\}} t_j^-$
MaxPW$^+$	PW$^+$ = $\sum_{j \in F_i^* \cup \{i\}} t_j^+$

Fonte: Adaptado de MOREIRA e RITT (2019).

ados a relações de precedência. É necessário destacar que na literatura a regra MaxPW$^-$ é conhecida por produzir resultados consideravelmente superiores às outras regras de seleção elaboradas manualmente (MOREIRA et al., 2012), (BORBA; RITT, 2014).

4.5 Componentes de regras de seleção

Ao analisar funções de prioridade de regras para seleção de tarefas, é possível identificar que existem certos componentes comuns a estas funções. Por exemplo, a função Time pode ser descrita como a soma sobre o conjunto de tarefas $\{ i \}$ do tempo de execução do trabalhador w. IF pode ser descrita como a soma sobre o conjunto $\{ i \}$ do atributo $|F_i|$, mas também pode ser vista como a soma sobre o conjunto F_i da constante 1, raciocínio análogo pode ser feito para a função F. PW$^-$ consiste na soma do valor t_i^- sobre o conjunto de tarefas $F_i^* \cup \{ i \}$, e PW$^+$ mantém este mesmo conjunto de tarefas, apenas muda o atributo para t_i^+. Por fim, a função Rank pode ser vista como a soma da constante 1 sobre um conjunto de trabalhadores: $\{ w' \in U_w \mid t_{w'i} < t_{wi} \}$.

De certa forma, funções de prioridade podem ser descritas como somas de atributos ou valores constantes sobre conjuntos, e o que diferencia uma função da outra são quais atributos, constantes ou conjuntos estão sendo usados. FUKUNAGA (2008) apresenta uma constatação semelhante ao analisar regras de seleção de variáveis em algoritmos heurísticos para o problema SAT, e chama estes elementos comuns a regras de *building blocks* (blocos de construção), termo que também usaremos aqui para nos referir a conjuntos, atributos ou constantes que compõem funções de prioridade de tarefas.

5 UMA LINGUAGEM PARA DESCREVER FUNÇÕES DE PRIORIDADE

Após identificar building blocks que compõem funções de prioridade de tarefas, desenvolvemos uma linguagem capaz de representar diferentes building blocks e combiná-los utilizando operações. Um programa nesta linguagem é uma função que computa a prioridade de uma única tarefa, com base nos valores de seus atributos.

Nossa linguagem é gerada por uma gramática livre de contexto. Nas próximas seções, apresentamos fundamentações teóricas que influenciaram o projeto da linguagem, os seus elementos, assim como a gramática elaborada.

5.1 Princípios para o projeto de funções de prioridade

Os operadores da nossa linguagem foram desenvolvidos com base nos quatro princípios gerais para o projeto de funções de prioridade propostos por OTTO e OTTO (2013), chamados de *agregação*, *combinação*, *especificidade estrutural* e *influência randômica*.

5.1.1 Agregação

O princípio da agregação afirma que arredondar ou discretizar valores de prioridade pode levar a resultados superiores. Isso ocorre porque valores de prioridade podem conter alguma forma de ruído e também porque é difícil determinar qual é a melhor escolha quando diferentes elementos possuem prioridades muito próximas. Por exemplo, se duas tarefas possuem prioridades 999 e 998 em uma execução onde valores de prioridade tendem a ser bem menores, é provável que o fato das duas tarefas terem um valor *alto* de prioridade seja mais informativo do que essa diferença de uma unidade nos valores.

O arredondamento ou discretização tem por objetivo causar empates de prioridades que são muito próximas, de forma que estes empates devem ser resolvidos por outras regras de prioridade, na esperança de que estas outras regras consigam diferenciar melhor qual escolha deve ser tomada no momento.

5.1.2 Combinação

A combinação afirma que o uso de diferentes formas de informação por uma regra pode produzir resultados superiores a utilizar apenas uma única forma de informação. Por exemplo, regras com funções de prioridade que utilizam tanto informação estrutural quanto temporal (ex. MaxPW^{-}) conseguem considerar em suas decisões informações que regras baseadas apenas em um tipo de atributo (ex. MaxIF ou MinTime) não conseguem.

OTTO e OTTO (2013) descreve que o princípio da combinação pode ser aplicado usando somas ponderadas de diferentes funções de prioridade, ou usando regras distintas para resolver empates. Um exemplo seria usar MinTime como regra de seleção, e caso a informação temporal não consiga discriminar bem quais tarefas são mais prioritárias, um empate poderia ser resolvido por uma regra como MaxF, que usa informação de outra natureza.

5.1.3 Especificidade estrutural

A especificidade estrutural é a observação de que certas regras possuem vantagens ou desvantagens dependendo de características do grafo D de precedências. Por exemplo, em uma instância onde não existem precedências entre tarefas ($A = \emptyset$), uma regra como MaxIF não consegue fazer distinção nenhuma entre tarefas.

OTTO e OTTO (2013) utiliza uma medida para caracterizar o grau de conectividade do grafo de precedências. Esta medida é chamada de *order strength* (OS). É sugerido combinar diferentes funções de prioridade, de forma que a função que de fato será usada depende do valor de OS do grafo residual atual. Sendo $U_t \subseteq T$ o conjunto de tarefas que ainda não foram alocadas a uma estação já adicionada à solução, OS é definido como:

$$\text{OS} = \frac{\sum_{i \in U_t} |F_i^*|}{\binom{|U_t|}{2}}$$

5.1.4 Influência randômica

O princípio da influência randômica baseia-se na observação de que pequenas alterações na execução de um método heurístico podem afetar consideravelmente o resultado produzido. Este princípio sugere executar repetidas vezes um algoritmo sobre determi-

nada instância, de forma que a cada vez sejam introduzidas randomicamente pequenas variações na execução (como por exemplo, com uma dada probabilidade selecionar uma tarefa disponível aleatória, ao invés da mais prioritária) e ao final seja escolhida a melhor solução produzida pelas execuções. Uma forma simples de implementar estas variações randômicas é a utilização de operadores com elementos probabilísticos para compor funções de prioridade de tarefas.

5.2 Elementos

A linguagem elaborada possui quatro elementos principais: constantes, atributos de tarefas, conjuntos de tarefas e operadores. Constantes são usadas como pesos em somas ponderadas, probabilidades em operadores estocásticos e fatores de arredondamento. Seguindo OTTO e OTTO (2013), mantivemos um conjunto finito de valores constantes em nossa linguagem, que são mostrados na Tabela 5.1.

Além de atributos já presentes em regras da literatura, complementamos o conjunto de atributos de tarefas variando operações de máximo, mínimo e soma sobre tempos de execução associados a trabalhadores disponíveis, considerando ou não o tempo do trabalhador da estação atual. Os atributos da nossa linguagem são explicados na Tabela 5.2.

Nossa linguagem possui operadores unários e binários, que são explicados na Tabela 5.3. Alguns operadores foram elaborados tendo em mente os quatro princípios descritos na Seção 5.1. Há operadores de arredondamento, seleção randômica, combinações convexas e ponderadas, soma sobre conjuntos de tarefas, entre outros. Os conjuntos de tarefas presentes são o conjunto de sucessores imediatos e conjunto total de sucessores, como já descritos, notados em nossa linguagem por IF e F respectivamente, e são utilizados pelo operador de soma sobre conjuntos (TSUM).

Tabela 5.1 – Constantes.

Tipo	Valores
Peso w	100, 10, 5, 2, 1, 0.5, 0.2, 0.1, 0.01
Probabilidade p	0.1, 0.3, 0.5, 0.7, 0.9
Fator de arredondamento α	0.01, 0.033, 0.1, 0.33

Fonte: MOREIRA e RITT (2019).

Tabela 5.2 – Atributos de tarefas.

Nome	Valor	Descrição
Time	t_{wi}	Tempo da tarefa i para o trabalhador atual w.
MaxTIC	$\max_{w' \in U_w} t_{w'i}$	Maior tempo de execução dentre trabalhadores disponíveis.
MaxTEC	$\max_{w' \in U_w - \{w\}} t_{w'i}$	Maior tempo de execução dentre trabalhadores disponíveis, sem w.
MinTIC	$\min_{w' \in U_w} t_{w'i}$	Menor tempo de execução dentre trabalhadores disponíveis.
MinTEC	$\min_{w' \in U_w - \{w\}} t_{w'i}$	Menor tempo de execução dentre trabalhadores disponíveis, sem w.
SumTIC	$\sum_{w' \in U_w} t_{w'i}$	Soma dos tempos de execução de trabalhadores disponíveis.
SumTEC	$\sum_{w' \in U_w - \{w\}} t_{w'i}$	Soma dos tempos de execução de trabalhadores disponíveis, sem w.
Rank	$\lvert\{w' \in U_w \mid t_{w'i} < t_{wi}\}\rvert$	Número de trabalhadores disponíveis com tempo menor que t_{wi}.
IF	$\lvert F_i \rvert$	Número de sucessores imediatos.
F	$\lvert F_i^* \rvert$	Número total de sucessores.

Fonte: MOREIRA e RITT (2019).

Funções de prioridade são representadas como árvores sintáticas válidas de acordo com as regras da nossa gramática e são avaliadas de forma recursiva para computar um único valor real de prioridade para uma tarefa. As entradas para uma função são os atributos da tarefa, localizados nas folhas das árvores, e o order strength do grafo residual de precedências, utilizado pelos operadores OS e OS*.

Para facilitar a notação, na Tabela 5.3 usamos f para se referir à expressão filha de operadores unários, l e r aos filhos à esquerda e à direita de operadores binários, o representa o order strength da instância residual atual e i uma tarefa qualquer sobre a qual está sendo calculado o valor de prioridade.

Tabela 5.3 – Operadores.

Nome	Valor	Descrição
	Operadores unários	
INV f	$-f(i)$	Multiplicação por -1.
TSUM S f	$\sum_{j \in S \cup \{i\}} f(j)$	Soma sobre conjunto de tarefas. $S \in \{\,\text{F, IF}\,\}$.
ROUND α f	$\lceil f(i)/(\alpha C) \rceil$	Arredondamento por αC.
	Operadores binários	
ADD l r	$l(i) + r(i)$	Soma.
SUB l r	$l(i) - r(i)$	Diferença.
MULT l r	$l(i)r(i)$	Multiplicação.
DIV l r	$l(i) \div r(i)$	Divisão.
RND p l r		Com probabilidade p retorna $l(i)$. Com probabilidade $(1-p)$ retorna $r(i)$.
OS l r	$o\,l(i) + (1-o)\,r(i)$	Combinação convexa utilizando order strength o.
OS* l r		Com probabilidade o retorna $l(i)$. Com probabilidade $(1-o)$ retorna $r(i)$.
CMB w l r	$wl(i) + (1-w)r(i)$	Combinação convexa com peso w.
CMB* w l r	$w'l(i) + (1-w')r(i)$	Combinação convexa com peso randômico $w' \sim U[w/1.1, 1.1w]$.
WCMB w l r	$wl(i) + r(i)$	Combinação ponderada com peso w.
WCMB* w l r	$w'l(i) + r(i)$	Combinação ponderada com peso randômico $w' \sim U[0.2w, 5w]$.
MIN l r	$\min\{l(i), r(i)\}$	Mínimo.
MAX l r	$\max\{l(i), r(i)\}$	Máximo.

Fonte: Adaptado de MOREIRA e RITT (2019).

5.3 Gramática

A linguagem proposta é gerada pela seguinte gramática livre de contexto:

⟨node⟩ ::= ⟨attribute⟩
| ⟨weight⟩
| ⟨binop⟩ ⟨node⟩ ⟨node⟩
| ⟨weighted binop⟩ ⟨weight⟩ ⟨node⟩ ⟨node⟩
| `INV` ⟨node⟩
| `ROUND` ⟨rounding factor⟩ ⟨node⟩
| `RND` ⟨probability⟩ ⟨node⟩ ⟨node⟩
| `TSUM` ⟨task set⟩ ⟨node⟩

⟨attribute⟩ ::= `Time` | `MaxTIC` | `MaxTEC` | `MinTIC` | `MinTEC`
| `SumTIC` | `SumTEC` | `Rank` | `IF` | `F`

⟨binop⟩ ::= `ADD` | `SUB` | `MULT` | `DIV` | `MAX` | `MIN` | `OS` | `OS*`

⟨weighted binop⟩ ::= `CMB` | `CMB*` | `WCMB` | `WCMB*`

⟨weight⟩ ::= `100` | `10` | `5` | `2` | `1` | `0.5` | `0.2` | `0.1` | `0.01`

⟨rounding factor⟩ ::= `0.01` | `0.033` | `0.1` | `0.33`

⟨probability⟩ ::= `0.1` | `0.3` | `0.5` | `0.7` | `0.9`

⟨task set⟩ ::= `IF` | `F`

5.4 Regras da literatura como programas na linguagem proposta

Se valores de prioridade forem maximizados, as regras da literatura apresentadas na Tabela 4.1 podem ser facilmente representadas como programas na linguagem proposta, como mostrado na Tabela 5.4.

Tabela 5.4 – Regras da Tabela 4.1 como programas.

Regra	Programa
MaxF	(F)
MaxIF	(IF)
MinTime	(INV (Time))
MinRank	(INV (Rank))
MaxPW^{-}	(TSUM F (MinTEC))
MaxPW^{+}	(TSUM F (MaxTEC))

Fonte: MOREIRA e RITT (2019).

6 ALGORITMOS GENÉTICOS E PROGRAMAÇÃO GENÉTICA

Neste capítulo, serão apresentados conceitos sobre algoritmos genéticos e também sobre a técnica de computação evolutiva chamada *Programação Genética*, que consiste na utilização de um algoritmo genético para evoluir programas de computador, com a finalidade de obter de forma automática um programa que resolva de forma eficaz um dado problema de interesse.

6.1 Algoritmos genéticos

Um algoritmo genético é uma meta-heurística usada para encontrar soluções para problemas de otimização, inspirada no processo biológico da evolução. O algoritmo mantém um conjunto de elementos chamados de *indivíduos*, que representam soluções para um dado problema, e cada indivíduo possui um valor associado chamado de *fitness* (aptidão), que em geral é uma medida de quão bem a solução representada pelo indivíduo resolve o problema de interesse.

A cada iteração, um *mecanismo de seleção* baseia-se no valor de fitness para escolher uma quantidade determinada de indivíduos da população, de forma que a probabilidade de um indivíduo ser selecionado é proporcional ao seu fitness. O algoritmo então aplica uma operação de *recombinação*, que combina características estruturais dos indivíduos selecionados para produzir novos indivíduos, os *descendentes*.

Para cada descendente gerado, uma pequena modificação é realizada em sua estrutura por uma *mutação*, aplicada de acordo com uma probabilidade. Depois de avaliar o fitness dos novos indivíduos, o algoritmo os insere na população seguindo uma *política de substituição* e inicia uma nova iteração. Este processo é repetido até que um *critério de parada* seja satisfeito.

A estratégia do algoritmo genético é baseada na hipótese de que a seleção e recombinação de indivíduos baseada no fitness tem o potencial de produzir novos indivíduos com qualidade superior aos seus ascendentes, e juntamente com a estratégia de substituição – que resulta na remoção de indivíduos de baixa qualidade da população, faz com que o fitness médio da população cresça ao longo da execução do algoritmo.

6.2 Programação genética

Programação genética é uma técnica de computação evolutiva que tem como ideia base a aplicação de um algoritmo genético onde os indivíduos são programas de computador. Em programação genética, um algoritmo genético mantém uma população de programas normalmente representados como árvores sintáticas, que são executados sobre um conjunto de entradas para definir o seu fitness.

POLI, LANGDON e MCPHEE (2008) cita uma série de casos de sucesso utilizando programação genética e destaca que esta técnica consegue produzir resultados competitivos com soluções elaboradas por humanos em diversas aplicações.

A representação de indivíduos como árvores possibilita a criação de várias estratégias de recombinação e mutação. Neste trabalho, iremos apresentar operações de recombinação e mutação baseadas nas ideias de FUKUNAGA (2008), POLI, LANGDON e MCPHEE (2008), OTTO e OTTO (2013).

7 UM ALGORITMO GENÉTICO PARA EVOLUIR FUNÇÕES DE PRIORIDADE

Este capítulo descreve as estratégias usadas na implementação de um algoritmo genético para a evolução de programas de computador, onde estes programas são funções de prioridade de tarefas codificadas na linguagem proposta no Capítulo 5. O objetivo é obter de forma automática uma função de prioridade que ajude métodos heurísticos para o ALWABP-2 a aprimorar a qualidade de soluções produzidas.

7.1 Função de fitness

Utilizaremos a heurística construtiva descrita em MOREIRA et al. (2012), cujo pseudocódigo foi mostrado no Algoritmo 1, para calcular o fitness de indivíduos. O fitness f_i de um indivíduo i com função de prioridade P_i é definido como a performance da heurística construtiva, usando como regra de seleção de tarefas $\max P_i$, aplicada a um conjunto I instâncias com soluções ótimas conhecidas propostas na literatura do ALWABP (CHAVES; LORENA; CRISTÓBAL, 2007).

Seja k uma instância do ALWABP, $C(k, P)$ o tempo de ciclo encontrado pelo método heurístico para a instância k utilizando $\max P$ como regra para seleção de tarefas e C_k^* o tempo de ciclo ótimo para a instância k. Definimos então o *desvio relativo* de $C(k, P)$ com relação a C_k^* como $R(k, P) = (C(k, P) - C_k^*)/C_k^*$. Dessa forma definimos o fitness f_i de um indivíduo i como a média do desvio relativo ao aplicá-lo sobre o conjunto de instâncias I: $f_i = \sum_{k \in I} R(k, P_i)/|I|$.

7.1.1 Avaliação em dois passos

Para um conjunto I com um número grande de instâncias, a execução do método heurístico sobre todas as instâncias do conjunto tende a se tornar custosa quando aplicada repetidamente, para o cálculo do fitness de cada indivíduo produzido pelo algoritmo genético, o que faz com que a busca do algoritmo no espaço de soluções se torne lenta.

Como forma de acelerar a busca do algoritmo genético no espaço de programas, a avaliação em dois passos primeiro avalia cada indivíduo em um subconjunto de instâncias $I' \subseteq I$, definindo um *fitness substituto* f'_i para o indivíduo i, usando a mesma métrica de média do desvio relativo.

Seja f'_i o fitness substituto calculado para um indivíduo i, e f^*_s o melhor fitness substituto presente na população no momento. A estratégia procede para o cálculo do fitness completo de i se e somente se o seu fitness substituto não é maior do que $f^*_s + \tau$, onde τ é uma tolerância definida como um parâmetro do algoritmo. Lembrando aqui que no nosso caso o fitness dos indivíduos é uma média sobre o desvio relativo com relação às soluções ótimas, e quanto menor for a média melhor é o desempenho do indivíduo.

A avaliação em dois passos considera que um indivíduo com baixo fitness substituto tende a ter um baixo fitness completo. Para isso, é necessário selecionar um subconjunto $I' \subseteq I$ que seja representativo para ajudar a indicar a qualidade de um indivíduo. A operação de seleção utiliza o fitness substituto para comparar indivíduos, usamos o fitness completo sobre todas instâncias apenas para reportar indivíduos de alta qualidade. Nos experimentos computacionais, apresentamos resultados com diferentes valores de tolerância τ.

7.2 Geração da população inicial

A população inicial é composta por indivíduos gerados randomicamente utilizando a gramática definida. Para cada indivíduo a ser adicionado à população inicial, aplicamos uma estratégia de aprimoramento, similar ao torneio usado na seleção: geramos k_0 indivíduos, computamos o fitness substituto para cada um destes indivíduos e adicionamos de fato à população aquele que tiver melhor fitness substituto, sendo k_0 um parâmetro do algoritmo genético. Esta estratégia tem a finalidade de aprimorar a qualidade da população inicial, com a intenção de que seja possível obter melhores soluções a longo prazo partindo desta população.

Também garantimos que árvores da população inicial tenham uma altura máxima permitida, para evitar o surgimento de árvores exageradamente grandes, tentando manter as funções simples e de rápida avaliação.

7.3 Seleção

Na operação de seleção, seguimos a recomendação de POLI, LANGDON e MCPHEE (2008) e aplicamos a estratégia de torneio para selecionar os dois indivíduos da população que serão usados na recombinação. Para selecionar o primeiro indivíduo, sorteamos com

probabilidade uniforme k_1 indivíduos da população atual, e o indivíduo sorteado com o melhor fitness substituto será o escolhido. Para o segundo indivíduo realizamos o mesmo processo, porém com k_2 indivíduos. Definimos k_1 e k_2 como parâmetros do algoritmo genético.

7.4 Recombinação

Como citado anteriormente, é possível criar diversas estratégias para combinação de árvores, de forma que a árvore resultante seja também um programa válido.

POLI, LANGDON e MCPHEE (2008) descreve um conjunto de operações clássicas de recombinação e mutação em programação genética. Utilizamos a mais comumente aplicada para recombinação, chamada de *recombinação de sub-árvores*. A recombinação de sub-árvores consiste em selecionar de forma randômica e independente dois pontos de recombinação (nodos) n_1 e n_2, um em cada pai p_1 e p_2. Como resultado é gerado um único indivíduo, formado a partir de uma cópia de p_2, substituindo-se a sub-árvore enraizada no seu ponto de recombinação n_2 por uma cópia da sub-árvore enraizada no ponto n_1 escolhido em p_1. É possível usar uma versão onde dois indivíduos novos são gerados desta forma, porém não é comum.

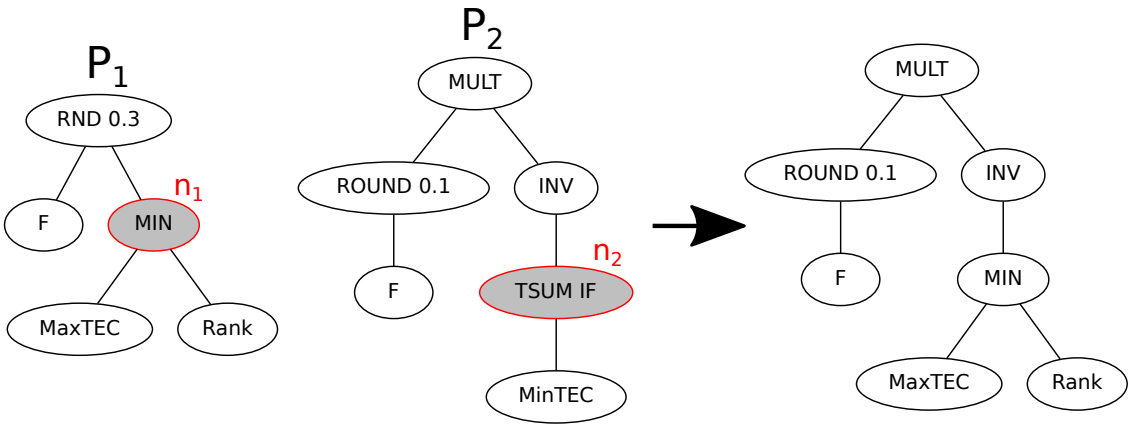


Figura 7.1 – Recombinação de sub-árvores.
Fonte: O Autor.

FUKUNAGA (2008) enfatiza que a operação de recombinação é a maior diferença entre sua aplicação para evoluir heurísticas para SAT e programação genética tradicional. Sua estratégia baseia-se no funcionamento de heurísticas conhecidas para o SAT e consiste na geração de novas árvores tendo como raiz operadores binários dentre um conjunto específico selecionado de sua linguagem, usando as árvores do primeiro e se-

gundo pai como filhos destes operadores binários. Selecionamos portanto 7 operadores da nossa linguagem com base nos princípios descritos na Seção 5.1, e geramos um total de 16 indivíduos combinando árvores p_1 e p_2 como descrito a seguir:

- (DIV p_1 p_2)
- (RND p p_1 p_2), para cada $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
- (WCMB w p_1 p_2), usando 6 pesos randômicos $w \in \{100, 10, 5, 2, 1, 0.5, 0.2, 0.1, 0.01\}$. Com probabilidade de 0.5, utilizamos o operador WCMB* ao invés de WCMB.
- (OS p_1 p_2)
- (OS* p_1 p_2)
- (MIN p_1 p_2)
- (MAX p_1 p_2)

7.5 Mutação

Em nosso estudo, realizamos testes utilizando quatro formas de mutação. Duas das formas foram elaboradas por nós, as quais nomeamos *mutações de expressão*. A terceira forma de mutação é comum na programação genética e chama-se *mutação de sub-árvores*. Ao final, aplicamos uma quarta forma de mutação proposta por FUKUNAGA (2008), chamada de *mutação por profundidade*. A seguir serão explicadas cada uma destas estratégias.

É necessário destacar que as mutações de expressão e a mutação de sub-árvores são realizadas em cópias de indivíduos e não nos indivíduos diretamente. Isso é feito para que o indivíduo original não seja perdido, de forma que ele ainda irá competir contra os novos indivíduos gerados por mutação para entrar na população. A mutação por profundidade, por outro lado, é aplicada diretamente (in-place) em todos os indivíduos resultantes após as operações de recombinação e mutações de expressão/sub-árvores.

7.5.1 Primeira mutação de expressões

A primeira forma de mutação de expressões foi criada com base no princípio da agregação descrito na Seção 5.1. Dado um indivíduo que tem uma árvore P, esta forma de mutação gera um novo indivíduo com árvore (ROUND α P), onde α é um fator de arredondamento escolhido randomicamente do conjunto de valores $\{ 0.01, 0.033, 0.1, 0.33 \}$.

7.5.2 Segunda mutação de expressões

A segunda mutação de expressões gera uma nova árvore P' a partir de uma árvore P, fazendo uma cópia de P e percorrendo seus nodos, de forma que cada nodo n é substituído por (INV n) com probabilidade $1/m$, onde m é a quantidade original de nodos na árvore P. Desta forma, a quantidade esperada de substituições é 1, e se nenhuma substituição ocorre, fazemos com que P' := (INV P).

O raciocínio que motivou esta operação é o fato de que sempre maximizamos as prioridades de tarefas dentro do algoritmo genético, porém existem funções que teriam melhor desempenho se as prioridades fossem minimizadas. Estas funções podem ser tanto árvores inteiras quanto sub-árvores dentro de uma árvore maior.

7.5.3 Mutação de sub-árvores

A forma de mutação de sub-árvores aplicada é a mais comum em programação genética. Consiste em dada uma árvore P, escolher com probabilidade uniforme um ponto de mutação (nodo) n em P e substituir a sub-árvore enraizada em n por uma árvore gerada aleatoriamente.

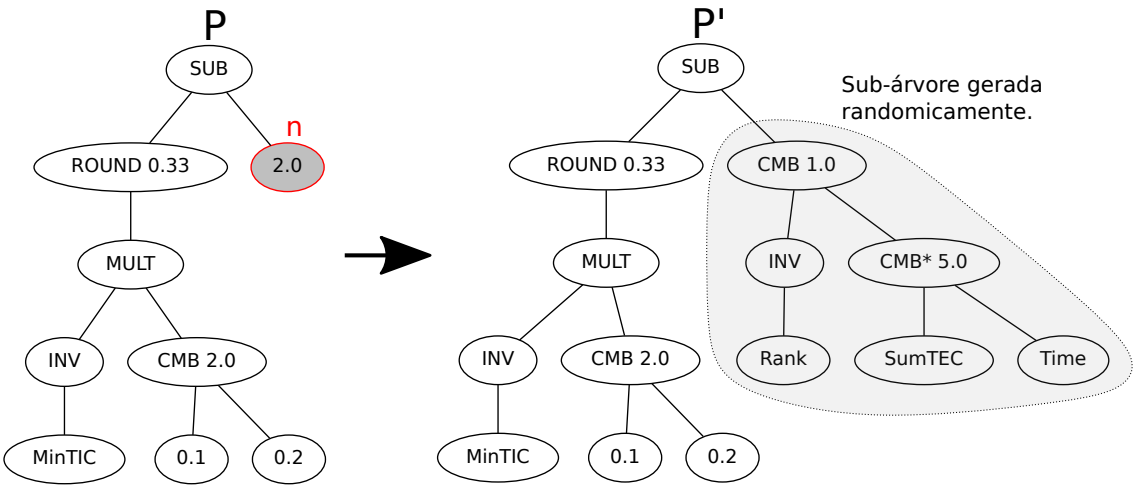


Figura 7.2 – Mutação de sub-árvores.
Fonte: O Autor.

7.5.4 Mutação por profundidade

Mutação por profundidade mantém uma altura máxima D permitida nas árvores da população. Quando uma árvore resultante de recombinação ou mutação de expressões/sub-árvores atinge uma altura superior a D, todo nodo não-folha desta árvore que tenha profundidade igual a D é substituído por uma folha gerada randomicamente (valor constante ou atributo de tarefa). A altura de uma árvore é o número de arestas no maior caminho de uma folha até a raiz, e a profundidade de um nodo é o número de arestas no caminho entre o nodo e a raiz. Além de servir como uma forma de mutação sobre as árvores, a mutação por profundidade evita o seu o crescimento ilimitado.

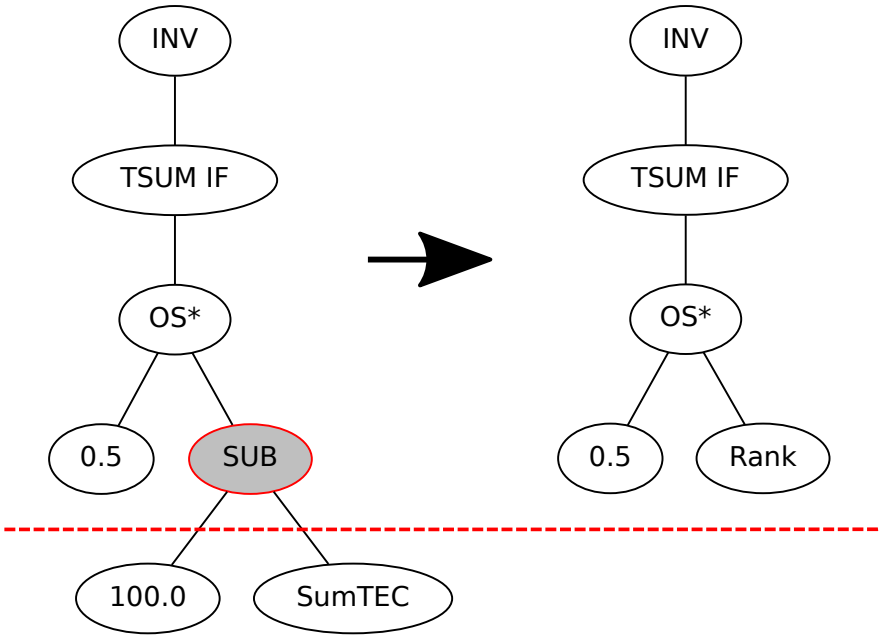


Figura 7.3 – Mutação por profundidade com altura máxima $D = 3$.
Fonte: O Autor.

7.6 Substituição da população

Seguindo a recomendação de POLI, LANGDON e MCPHEE (2008) utilizamos a estratégia de *steady-state* para realizar a substituição da população. Para cada indivíduo i resultante da aplicação de recombinação e/ou mutação, calculamos o seu fitness substituto f'_i, como definido na Seção 7.1.1. Seja $i^\dagger$ o indivíduo com pior fitness substituto atualmente presente na população. Se o indivíduo i tem um fitness substituto melhor do que o indivíduo $i^\dagger$, então i é inserido na população e $i^\dagger$ removido. Esta estratégia mantém

o tamanho da população constante e faz com que o fitness substituto médio da população tenda a crescer ao longo do tempo.

7.7 Critério de parada

Como critério de parada para o algoritmo genético implementado, utilizamos o tempo limite de execução do algoritmo, a fim de tornar mais conveniente a execução de testes e permitir a execução do mesmo por períodos prolongados.

7.8 Pseudocódigo

O Algoritmo 2 mostra o pseudocódigo do algoritmo genético implementado. A operação de recombinação é realizada com probabilidade p_c, e caso não ocorra, aplicamos uma mutação sobre o primeiro pai escolhido. Para cada descendente resultante da recombinação aplicamos a mutação com probabilidade p_m. A operação de *poda* corresponde à mutação por profundidade, e *avalia_e_substitui* é a função que calcula o fitness substituto do indivíduo e compara com o pior e melhor fitness substitutos da população atual, para determinar se o indivíduo será inserido ou não na população e se seu fitness completo será calculado para que seja reportado nos logs do algoritmo.

Algoritmo 2: Algoritmo genético para evoluir funções de prioridade.

```
1  início
2      população ← Inicializa(k_0, tamanho_pop)
3      enquanto critério de parada não satisfeito faça
4          pai_1 ← torneio(k_1, população)
5          pai_2 ← torneio(k_2, população)
6          descendentes ← ∅
7          mutantes ← ∅
8          com probabilidade de recombinação p_c faça
9              descendentes ← recombinação(pai_1, pai_2)
10             para cada i ∈ descendentes faça
11                 com probabilidade de mutação p_m faça
12                     mutantes ← mutantes ∪ mutação(i)
13         senão
14             mutantes ← mutação(pai_1)
15         podados ← poda(descendentes ∪ mutantes)
16         para cada i ∈ podados faça
17             avalia_e_substitui(i, população)
```

8 ESTRATÉGIAS APLICADAS AO ALWABP

Durante nossa análise, identificamos algumas estratégias que podem ser aplicadas na tentativa de melhorar decisões tomadas por métodos heurísticos para o ALWABP-2. Estas estratégias são usadas principalmente após a detecção de certas situações onde determinadas tarefas só poderão ser realizadas por algum trabalhador em específico, de forma que sem a identificação de tais situações, a heurística por vezes toma decisões que impossibilitam a construção de uma solução viável. Abaixo, explicamos as estratégias utilizadas e a motivação por trás de cada uma delas.

8.1 Pré-seleção

Durante a execução de um SoAP, pode ocorrer que para uma certa tarefa ainda não realizada $i \in T$ exista apenas um trabalhador disponível $w \in U_w$ que possa realizar i. Isso ocorre quando existe apenas um trabalhador na instância que consegue fazer a tarefa em tempo $\leq C$, ou quando sobram exatamente dois trabalhadores que podem realizar a tarefa e um desses trabalhadores é alocado para uma estação adicionada à solução, restando apenas ao outro a realização da tarefa.

A pré-seleção identifica este tipo de situação, onde torna-se necessário que um determinado trabalhador w realize uma certa tarefa i para que seja possível construir uma solução viável. Quando isso ocorre, marcamos a tarefa i como *reservada* para o trabalhador w.

Seja R_w o conjunto de todas as tarefas reservadas para o trabalhador w no momento atual da execução da heurística. Toda vez que uma estação de trabalho começar a ser construída para o trabalhador w, a pré-seleção tenta primeiro alocar nesta estação todas as tarefas ainda não realizadas da união $\bigcup_{i \in R_w} P_i^* \cup \{ i \}$, *antes* de começar a selecionar tarefas por prioridade para a estação. Se não for possível alocar todas estas tarefas à estação de w sem que o workload ultrapasse C, então w não pode ser selecionado neste momento para compor a solução, pois isso impossibilitaria a construção de uma solução viável.

O objetivo da pré-seleção é ajudar regras de seleção que não conseguem identificar tarefas não alocadas que devem ser realizadas por um único trabalhador em específico (como por exemplo regras MinTime, MaxF, MaxIF, MaxPW$^+$ e MinRank). É interessante observar que se tempos de execução superiores a C forem transformados em ∞ antes

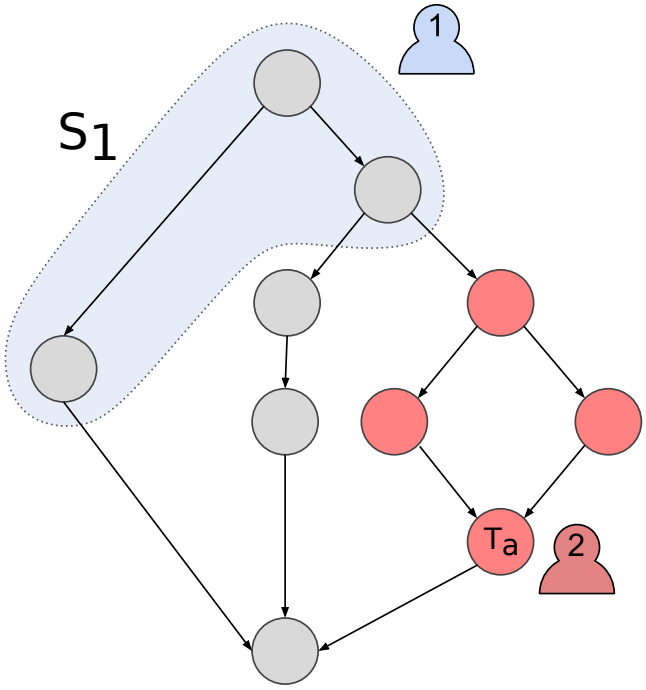


Figura 8.1 – Pré-seleção.
Fonte: O Autor.

de executar o SoAP então a regra MaxPW^{-}, que tem o melhor desempenho dentre as regras manualmente elaboradas para o ALWABP, consegue mesmo sem a pré-seleção identificar tarefas que devem necessariamente ser feitas pelo trabalhador da estação atual e tenta alocar elas e seus predecessores primeiro, visto que estas tarefas e todos os seus predecessores terão valor de prioridade ∞.

A Figura 8.1 mostra um caso de pré-seleção. Caso a tarefa T_a possa ser realizada apenas pelos trabalhadores 1 e 2 e o trabalhador 1 seja alocado à estação S_1 da solução, então T_a será reservada para o trabalhador 2, e para que ele possa concorrer para ser adicionado à próxima estação da solução, deve pelo menos realizar todas as tarefas marcadas em vermelho.

8.2 Redução do cone

A redução do cone é uma outra estratégia lógica relacionada à pré-seleção. Seu objetivo porém é modificar tempos de execução de tarefas ainda não realizadas, com o intuito de ajudar regras de seleção de tarefas e trabalhadores.

Dadas duas tarefas $i, j \in T$ tais que $j \in F_i^*$, definimos como *bicone* B_{ij} o conjunto de tarefas na intersecção $F_i^* \cap P_j^*$. Se as tarefas i e j estiverem reservadas para um

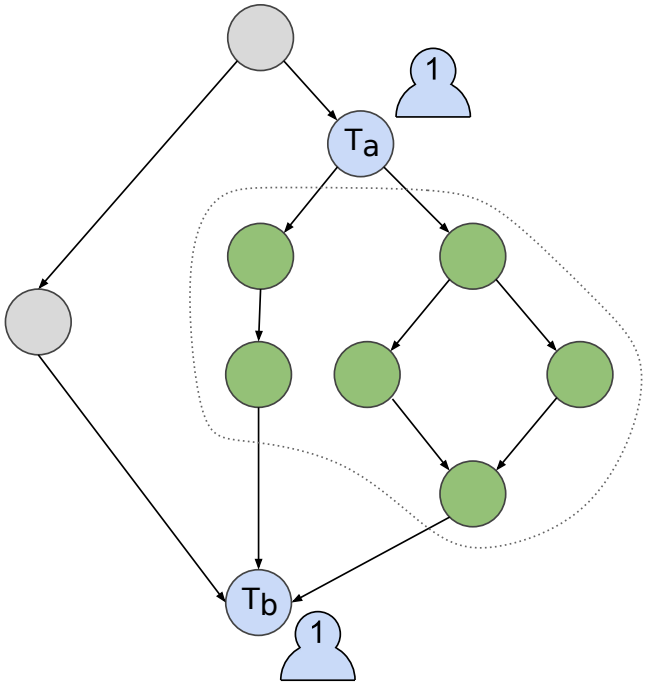


Figura 8.2 – Redução do cone.
Fonte: O Autor.

trabalhador w então elas só podem ser realizadas por w, e por causa da restrição de que as relações de precedência entre tarefas devem ser respeitadas na solução, isso implica que toda tarefa em B_{ij} deve ser realizada por w também, mesmo que exista um trabalhador $w' \in U_w$ para o qual existe $k \in B_{ij} \mid t_{w'k} \leq C$.

Nesse caso, a redução do cone consiste em para toda tarefa em B_{ij} atribuir para ∞ o tempo de execução associado a trabalhadores diferentes de w. Esta modificação não altera a factibilidade da instância residual atual e tem a finalidade de tornar as decisões da heurística mais inteligentes, permitindo que regras de seleção de tarefas e trabalhadores identifiquem que tarefas neste bicone devem ser realizadas apenas pelo trabalhador w.

A Figura 8.2 ilustra um caso de redução do cone. Se as tarefas T_a e T_b forem reservadas para o trabalhador 1, então ele deve realizar todas as tarefas marcadas em verde também. Nesse caso, atribuímos o tempo dos outros trabalhadores nas tarefas em verde para ∞.

8.3 Normalizações

Alguns atributos de tarefas apresentam intervalos de valores que em geral diferem de outros atributos. Por exemplo, SumTIC vai ser maior que Time em praticamente todos

Tabela 8.1 – Normalizações.

Atributo	Coeficiente de normalização
Time	C
MaxTIC	C
MinTIC	C
MaxTEC	C
MinTEC	C
SumTIC	$\|U_w\|C$
SumTEC	$\|U_w\|C$
Rank	$\|U_w\|$
F	$\|U_t\| + 1$
IF	$\delta_{max} + 1$
Conjunto de tarefas (TSUM)	**Coeficiente de normalização**
F	$\sqrt[3]{\|U_t\|} + 1$
IF	$\sqrt{\delta_{max}} + 1$

Fonte: MOREIRA e RITT (2019).

os casos, assim como F tende a ser maior do que IF ou Rank. Somas sobre conjuntos de tarefas, realizadas pelo operador TSUM, também tendem a ter valores maiores do que valores calculados sobre uma única tarefa.

Estas diferenças de intervalos fazem com que determinados nodos dominem o valor de uma função de prioridade, necessitando portanto que o algoritmo genético evolua somas ponderadas adequadas para que nodos com valores menores consigam contribuir de forma significativa no calculo de prioridades.

Por este motivo, nós aplicamos normalizações sobre atributos de tarefas e também sobre o resultado de somas sobre conjuntos, conforme descrito na Tabela 8.1. A normalização consiste na divisão do valor do atributo ou soma por um coeficiente de normalização. As normalizações aplicadas baseiam-se no trabalho de OTTO e OTTO (2013). Seguindo a notação já utilizada, U_w representa o conjunto de trabalhadores disponíveis, U_t o conjunto de tarefas ainda não realizadas, C o tempo de ciclo atual e $\delta_{max} = \max_{i \in U_t}\{|F_i|\}$

8.4 Limitar tempos de execução

Durante a execução de um SoAP para um limite superior C no tempo de ciclo, é evidente que uma tarefa i só pode ser alocada para um trabalhador w se $t_{wi} \leq C$. Com base nessa observação, e como tentativa de guiar o método heurístico a tomar decisões melhores, aplicamos a estratégia de atribuir ∞ para todo tempo de execução $t_{wi} > C$.

Esta modificação pode ajudar a heurística a identificar incompatibilidades entre tarefas e trabalhadores para o tempo de ciclo considerado, porém tende a causar um número maior de empates de tarefas com valor de prioridade ∞. Na seção de experimentos computacionais, mostramos resultados obtidos ativando e desativando esta estratégia de limitar tempos de execução.

9 RESULTADOS COMPUTACIONAIS

Para a realização de experimentos com o algoritmo genético, dividimos as operações de recombinação e mutação descritas nas Seções 7.4 e 7.5 em duas estratégias, chamadas de *estratégia Clássica* e *estratégia Alternativa*, para referenciar a origem das operações utilizadas. A estratégia Alternativa utiliza as operações propostas por FUKUNAGA (2008) para realizar recombinação, combinando árvores com operadores binários, e como mutação utiliza as mutações de expressão elaboradas. Já a estratégia Clássica baseia-se em POLI, LANGDON e MCPHEE (2008), usando como recombinação e mutação as operações de recombinação de sub-árvores e mutação de sub-árvores apresentadas. Em ambas as estratégias, aplicamos a operação de mutação por profundidade para limitar a altura máxima de árvores.

Todos os experimentos foram executados em um PC com sistema operacional Ubuntu 20.04, processador AMD Ryzen 9 3900X 12-Core 3.8 GHz e 32GB de memória RAM. A heurística construtiva descrita no Algoritmo 1, o algoritmo genético e o IPBS utilizados nos experimentos, assim como a gramática proposta, foram implementados na linguagem C++ 17 e compilados com GNU G++ 9.3.0 utilizando flag de otimização $-O3$.

Para gerar números pseudoaleatórios, utilizamos as funções *uniform_int_distribution* e *uniform_real_distribution* da biblioteca padrão do C++, junto com o gerador *Mersenne Twister 19937* (mt19937). Todas as execuções do algoritmo genético utilizaram o valor 1 como semente para o gerador de números pseudoaleatórios. Em execuções da heurística construtiva e do IPBS para avaliar funções descobertas, utilizamos os valores de 1 a n como sementes para o gerador, sendo n o número de replicações realizadas (3 para a calibração de parâmetros e 10 nos outros casos).

Para a regra de seleção de trabalhadores, nos baseamos em MOREIRA et al. (2012) e usamos a regra denominada MinRLB (Minimum Restricted Lower Bound) em todos os experimentos. Sendo w o trabalhador para o qual está sendo computado o valor de RLB, U_w o conjunto de trabalhadores disponíveis, U_t o conjunto de tarefas não realizadas, t_i^- o menor tempo de execução da tarefa i no conjunto de trabalhadores $U_w - \{ w \}$ e T_w o conjunto de tarefas na estação do trabalhador w, a regra MinRLB seleciona um trabalhador que tenha o menor valor de RLB, conforme a fórmula:

$$\text{RLB} = \frac{\sum_{i \in U_t - T_w} t_i^-}{|U_w| - 1}$$

9.1 Conjunto de instâncias

Usamos o conjunto de 320 instâncias criado por CHAVES, LORENA e CRISTÓBAL (2007), todas com soluções ótimas conhecidas. Este conjunto foi gerado a partir de quatro instâncias clássicas do SALBP chamadas de Heskia, Roszieg, Wee-mag e Tonge, propostas por HOFFMANN (1990), variando cinco características em dois níveis: número de tarefas $|T|$, order strength OS (Seção 5.1.3), número de trabalhadores $|W|$, tempo de execução de tarefas e o percentual de incompatibilidades entre tarefas e trabalhadores (tempos iguais a ∞). A Tabela 9.1 mostra os parâmetros de variação utilizados para a geração do conjunto de instâncias do ALWABP a partir das quatro instâncias do SALBP. t_i refere-se ao tempo original de execução da tarefa i na instância do SALBP.

O conjunto total de 320 instâncias é composto por 32 grupos de 10 instâncias que compartilham características em comum. Em nossos experimentos, realizamos a avaliação em dois passos conforme descrito na Seção 7.1.1 utilizando 32 instâncias para calcular o fitness do primeiro passo (fitness substituto), uma instância de cada grupo.

Tabela 9.1 – Variação de fatores para geração do conjunto de instâncias.

	Nível	
Fator	Baixo	Alto
Número de tarefas ($\|T\|$)	$25 - 28$	$70 - 75$
Order strength (OS)	$22\% - 23\%$	$59\% - 72\%$
Número de trabalhadores ($\|W\|$)	$\|T\|/7$	$\|T\|/4$
Variabilidade no tempo de tarefas	$[1, t_i]$	$[1, 2t_i]$
Percentual de incompatibilidades	10%	20%

Fonte: MOREIRA e RITT (2019).

9.2 Calibração de parâmetros

O conjunto de parâmetros do algoritmo genético e o domínio de cada parâmetro são descritos na Tabela 9.2. *Limitar tempos de execução* (λ) indica se será atribuído ∞ para tempos maiores que C, tolerância τ é usada para determinar se um indivíduo prosseguirá para o segundo passo de avaliação (Seção 7.1.1). D é a altura máxima permitida para árvores, k_0 o número de indivíduos gerados e avaliados no torneio para cada indivíduo a ser inserido na população inicial (Seção 7.2). Os números de indivíduos sorteados da população para selecionar o primeiro e o segundo pai para a recombinação são k_1 e k_2.

Tabela 9.2 – Parâmetros do algoritmo genético.

Parâmetro	Domínio	Valor padrão
Tamanho da população N	$\mathbb{N}$	–
Probabilidade de recombinação p_c	0–1	0.8
Probabilidade de mutação p_m	0–1	0.3
Limitar tempos de execução λ	V/F	–
Tolerância τ	0–100%	–
Altura máxima D	$\mathbb{N}$	6
Torneio população inicial k_0	$\mathbb{N}$	11
Torneio primeiro pai k_1	$\mathbb{N}$	2
Torneio segundo pai k_2	$\mathbb{N}$	2

Fonte: Adaptado de MOREIRA e RITT (2019).

Calibramos o tamanho da população N e os parâmetros λ e τ, pois estão relacionados a duas estratégias importantes descritas neste trabalho. Para o conjunto restante de parâmetros, atribuímos valores padrão conforme a Tabela 9.2. Estes valores foram obtidos em uma calibração de parâmetros previamente realizada em MOREIRA e RITT (2019).

Para definir a qualidade de um conjunto de valores de parâmetros, executamos o algoritmo genético por um tempo fixo de cinco horas e selecionamos para análise as 10 regras produzidas na execução que apresentaram menor média do desvio relativo sobre os tempos de ciclo ótimos das 320 instâncias. Para cada regra selecionada, executamos a heurística construtiva sobre todas as instâncias três vezes (visto que as regras podem ter elementos probabilísticos) obtendo portanto três médias do desvio relativo R_1, R_2 e R_3 para cada regra. A qualidade de uma regra é dada por $(R_1 + R_2 + R_3)/3$, a qualidade da configuração de parâmetros é avaliada conforme a média $\overline{f}$, o mínimo f_{min}, o máximo f_{max} e o desvio padrão f_σ da qualidade das 10 regras avaliadas geradas pelo genético utilizando a configuração. Observe que estamos usando médias do desvio relativo com relação a soluções ótimas para definir a qualidade de regras, portanto valores menores de f indicam melhor desempenho.

Seguindo MOREIRA e RITT (2019) utilizamos a estratégia de *one-factor-at-a-time* para realizar a calibração dos parâmetros. Variamos o parâmetro sendo calibrado, enquanto mantemos os outros constantes. Após as execuções do genético para diferentes valores do parâmetro, escolhemos o valor da configuração que obteve maior qualidade e fixamos este valor na calibração dos parâmetros seguintes. Para posterior comparação das estratégias Clássica e Alternativa, a calibração de parâmetros foi feita separadamente para as duas estratégias. As Tabelas 9.3 e 9.4 mostram os resultados da calibração para as

duas estratégias, na ordem em que foram calibrados os parâmetros, sendo destacado em vermelho o valor escolhido para ser fixado em cada parâmetro.

Tabela 9.3 – Calibração de parâmetros para a estratégia Clássica.

Parâmetro	Valor	$\overline{f}$ (%)	$f_{\min}$ (%)	$f_{\max}$ (%)	f_σ (%)
N	50	19.3	19.2	19.5	0.12
	150	19.4	18.9	19.8	0.26
	250	19.2	18.9	19.6	0.22
τ	**1%**	18.6	18.2	19.0	0.23
	3%	19.1	18.7	19.6	0.22
	5%	19.5	18.9	19.9	0.27
λ	V	19.2	18.9	19.4	0.15
	F	18.8	18.4	19.0	0.19

Fonte: O Autor.

Tabela 9.4 – Calibração de parâmetros para a estratégia Alternativa.

Parâmetro	Valor	$\overline{f}$ (%)	$f_{\min}$ (%)	$f_{\max}$ (%)	f_σ (%)
N	50	19.7	19.0	19.9	0.24
	150	18.6	18.3	19.0	0.19
	250	19.1	18.9	19.4	0.15
τ	1%	19.2	19.0	19.4	0.10
	3%	19.2	18.8	19.5	0.22
	5%	19.2	18.9	19.7	0.20
λ	**V**	18.9	18.9	19.0	0.04
	F	19.5	19.3	19.6	0.09

Fonte: O Autor.

9.3 Comparação com regras da literatura

Após calibrar os parâmetros para as estratégias Clássica e Alternativa, executamos o algoritmo genético durante 15 horas para cada estratégia. Pegamos a melhor regra encontrada por cada uma das duas execuções e rodamos 10 vezes a heurística construtiva aplicando cada regra sobre todas as instâncias. A Tabela 9.5 apresenta resultados sobre as execuções da melhor regra das duas estratégias (Clássica$*$ e Alternativa$*$) e também três das melhores regras da literatura. Note que para as regras da literatura (MaxPW$^-$, MaxPW$^+$ e MinRank), $\overline{f} = f_{min} = f_{max}$ e $f_\sigma = 0$, pois essas regras não possuem elementos probabilísticos (para a mesma regra, a heurística sempre produz o mesmo resultado).

Tabela 9.5 – Comparação de regras descobertas com as melhores regras da literatura, usando a heurística construtiva.

Regra	Direção	$\overline{f}$ (%)	f_{min} (%)	f_{max} (%)	f_σ (%)	$\overline{t}$ (s)	t_{min} (s)	t_{max} (s)	t_σ (s)
Alternativa$*$	N	19.0	18.6	19.5	0.24	3.07	3.00	3.12	0.04
	R	25.9	25.4	26.4	0.29	6.24	6.05	6.66	0.15
	BD	16.1	15.7	16.6	0.28	8.10	7.88	8.31	0.12
Clássica$*$	N	19.0	18.7	19.4	0.20	4.42	4.32	4.54	0.06
	R	26.9	26.4	27.4	0.33	9.32	8.99	9.64	0.21
	BD	16.3	15.9	16.7	0.22	12.52	12.04	12.95	0.30
MaxPW$^-$	N	20.7	20.7	20.7	0.0	1.98	1.96	2.00	0.01
	R	30.2	30.2	30.2	0.0	3.19	3.15	3.23	0.02
	BD	17.4	17.4	17.4	0.0	4.24	4.17	4.31	0.03
MaxPW$^+$	N	24.1	24.1	24.1	0.0	1.84	1.81	1.87	0.02
	R	32.9	32.9	32.9	0.0	2.68	2.67	2.72	0.01
	BD	21.1	21.1	21.1	0.0	3.60	3.55	3.67	0.04
MinRank	N	25.7	25.7	25.7	0.0	1.71	1.69	1.75	0.01
	R	28.7	28.7	28.7	0.0	2.24	2.22	2.28	0.01
	BD	20.7	20.7	20.7	0.0	3.04	2.99	3.11	0.03

Fonte: O Autor.

Para comparação, além dos valores calculados com base na média do desvio relativo f com relação às soluções ótimas, também é utilizado a média $\overline{t}$, mínimo t_{min}, máximo t_{max} e desvio padrão t_σ do tempo de execução da heurística, em segundos, sobre todo o conjunto de 320 instâncias.

A coluna *direção* refere-se a três estratégias de alocação que podem ser aplicadas dependendo de como se usa o grafo D de precedências. A estratégia *normal* (N) considera as precedências originais do grafo. *Reversa* (R) consiste em inverter as direções das precedências em D, executar a heurística, e depois inverter novamente as precedências entre as estações de trabalho na solução encontrada. *Bidirecional* (BD) primeiro executa a heurística para a direção normal, depois para reversa, e retorna a melhor solução dentre as duas. Existem casos onde a direção normal encontra uma solução melhor do que a reversa, e outros casos onde ocorre o contrário, por isso a ideia de executar em duas direções e retornar a melhor tende a produzir soluções superiores do que executar em uma direção única.

Na Tabela 9.5 é possível observar que a direção BD produziu resultados superiores para todas as regras, seguida da direção N e depois R. A alocação reversa necessita em geral de mais tempo para conseguir encontrar uma solução viável do que a alocação normal. O desvio padrão das médias do desvio relativo f para as regras Alternativa$*$ e Clássica$*$

são baixos (menos de 1%) e é possível notar que estas duas regras descobertas pelo algoritmo genético, apesar de terem um tempo de execução maior, produziram resultados melhores do que as regras da literatura, nas três direções de alocação. Alternativa$*$ conseguiu produzir resultados ligeiramente superiores a Clássica$*$, principalmente nas direções R e BD.

9.4 Aplicação em um IPBS

Nesta seção, comparamos as melhores regras descobertas pelas estratégias Alternativa e Clássica com as mesmas três regras da literatura anteriores, aplicadas por um método heurístico que faz parte do estado da arte para o ALWABP, a *Busca em feixe probabilística iterada* (IPBS, *Iterated Probabilistic Beam Search*) proposta por BORBA e RITT (2014). O IPBS também utiliza um SoAP no seu processo de busca por uma solução, porém tanto o SoAP quanto a estratégia de busca no espaço de tempos de ciclo C diferem da heurística construtiva do Algoritmo 1.

Para selecionar a tarefa disponível a adicionar na estação sendo construída, o IPBS não seleciona sempre a tarefa de maior (ou menor) prioridade, mas sim uma tarefa randomicamente, computando probabilidades de seleção com base em valores de prioridade. Durante o SoAP, o IPBS aplica uma busca em feixe mantendo um conjunto de soluções parciais para a instância, sendo que o número de soluções mantidas, chamado de *largura* do feixe e notado por γ, é um parâmetro do algoritmo. O que acontece é que para cada nova estação a ser adicionada para a solução, o IPBS constrói várias estações para cada um dos trabalhadores disponíveis, visto que a regra para seleção de tarefas para estações é probabilística. Dentre todas as estações construídas, o IPBS seleciona no máximo as γ melhores para compor o próximo feixe, de acordo com a regra de seleção de trabalhadores estabelecida. O número de estações construídas para cada trabalhador é chamado de *fator de ramificação* ϕ e também é um parâmetro do IPBS.

Como estratégia de busca no espaço de soluções, o IPBS mantém um limite inferior $\underline{C}$ estimado com base em tempos de execução de tarefas da instância e um limite superior $\overline{C}$ no tempo de ciclo, e realiza uma *busca por limite superior*, com base no tempo $\overline{C}$ atual, executando o SoAP para todo tempo de ciclo no intervalo $[\max\{\,\underline{C}, \lfloor\, p\,\overline{C}\,\rfloor\,\}, \overline{C}-1]$ para um fator $p \in (0,1)$ escolhido. Caso uma solução viável seja encontrada no intervalo, o limite superior $\overline{C}$ é atualizado e a busca por limite superior prossegue. Caso não seja encontrada solução viável, o IPBS continua procurando em tempos de ciclo no mesmo

intervalo, visto que o algoritmo possui elementos probabilísticos e uma nova execução sobre os mesmos tempos de ciclo pode vir a encontrar uma solução viável.

Três parâmetros são utilizados pela busca por limite superior realizada: o tempo mínimo de busca t_{min}, o tempo máximo de busca t_{max} e o número de repetições ζ. Caso solução viável não seja encontrada para um intervalo $[\max\{\,\underline{C}, \lfloor\, p\,\overline{C}\rfloor\,\}, \overline{C} - 1]$ durante a busca, o IPBS continua procurando no mesmo intervalo por no mínimo t_{min} segundos. O algoritmo termina quando a busca alcança e encontra solução para o limite inferior $\underline{C}$, ou quando para um mesmo intervalo passa t_{max} segundos ou realiza ζ tentativas sem encontrar solução viável. Ao terminar, a melhor solução viável encontrada é retornada.

A Tabela 9.6 apresenta resultados da execução das regras avaliadas usando o IPBS. Para os parâmetros do algoritmo seguimos a estratégia de BORBA e RITT (2014) e usamos $\gamma = 125$, $\phi = 5$, $p = 0.95$ e $\zeta = 20$. Devido ao elevado tempo de execução do algoritmo, usamos $t_{min} = 0$ e $t_{max} = 6$ segundos. Como o objetivo é comparar regras quando aplicadas a um dos melhores métodos existentes para o ALWABP-2, utilizamos apenas a alocação bidirecional para análise na Tabela 9.6, visto que esta forma de alocação em geral produz resultados bastante superiores às outras duas. É possível notar que o tempo de execução é muito superior ao tempo da heurística construtiva, porém a qualidade das soluções é muito superior também (uma queda de aproximadamente 16% para 3% acima da otimalidade). Observamos que neste teste a regra Clássica$*$ não conseguiu superar a melhor regra existente na literatura (MaxPW$^-$), porém a regra Alternativa$*$ conseguiu, apesar de apresentar tempo de execução superior.

Tabela 9.6 – Comparação de regras usando o IPBS.

Regra	$\overline{f}$ (%)	f_{min} (%)	f_{max} (%)	f_σ (%)	$\overline{t}$ (s)	t_{min} (s)	t_{max} (s)	t_σ (s)
Alternativa$*$	2.8	2.7	2.9	0.07	1504	1494	1512	6.20
Clássica$*$	3.3	3.2	3.4	0.08	1605	1594	1620	8.83
MaxPW$^-$	3.1	3.0	3.2	0.07	1347	1335	1360	7.99
MaxPW$^+$	4.1	4.0	4.2	0.04	1376	1345	1397	14.31
MinRank	3.5	3.4	3.7	0.09	1301	1284	1317	10.66

Fonte: O Autor.

9.5 A melhor regra

Para deixar registrado, e também para fins de curiosidade, apresentamos nessa seção a melhor regra de seleção de tarefas descoberta pelo algoritmo genético, dentre

todos os diversos testes e experimentos que foram realizados ao longo deste trabalho. Esta regra não foi descoberta por nenhum experimento apresentado anteriormente, mas sim por uma execução do algoritmo genético durante um teste utilizando a estratégia Clássica, que conseguiu convergir para uma solução melhor do que o usual.

A Figura 9.1 mostra a árvore desta melhor regra, e destacamos que ela combina diversos atributos de tarefas e operadores. É interessante observar que apesar de ser possível analisar a regra e tentar extrair alguma informação sobre como ela funciona, é muito difícil determinar *porque* essa regra em específico funciona tão bem, assim como é extremamente difícil tentar elaborar uma regra de seleção como essa manualmente sem tal certeza. Essa observação sustenta o fato de que a descoberta de regras eficientes de seleção para métodos heurísticos baseados em regras de prioridade é uma tarefa difícil para o ser humano, mas adequada para técnicas de inteligência artificial como programação genética.

A Tabela 9.7 apresenta o desempenho da melhor regra descoberta, quando aplicada pela heurística construtiva do Algoritmo 1 usando cada uma das direções de alocação, e também pelo IPBS usando a direção BD. Note que ela tem um desempenho melhor do que todas as regras apresentadas na Tabela 9.5 e o seu desvio padrão f_σ é menor do que Alternativa$*$ e Clássica$*$ nas três direções também, apesar dela apresentar tempos de execução um pouco superiores. Os valores apresentados na tabela também foram obtidos a partir de 10 execuções da heurística construtiva e do IPBS, usando a melhor regra descoberta, sobre todas as 320 instâncias.

Na comparação utilizando o IPBS, a melhor regra também tem desempenho superior a todas as regras apresentadas, com desvio padrão semelhante, porém o seu tempo de execução foi inclusive menor do que Alternativa$*$ e Clássica$*$.

Tabela 9.7 – Desempenho da melhor regra descoberta, aplicada pela heurística construtiva (HC) nas três estratégias de alocação, e pelo IPBS na estratégia bidirecional.

Algoritmo	Direção	$\overline{f}$ (%)	f_{min} (%)	f_{max} (%)	f_σ (%)	$\overline{t}$	t_{min}	t_{max}	t_σ
HC	N	18.5	18.1	18.8	0.19	4.58	4.43	4.68	0.07
	R	24.1	23.8	24.5	0.20	10.48	10.26	10.76	0.15
	BD	15.0	14.6	15.3	0.19	14.04	13.78	14.45	0.22
IPBS	BD	2.7	2.5	2.8	0.07	1464	1454	1471	5.61

Fonte: O Autor.

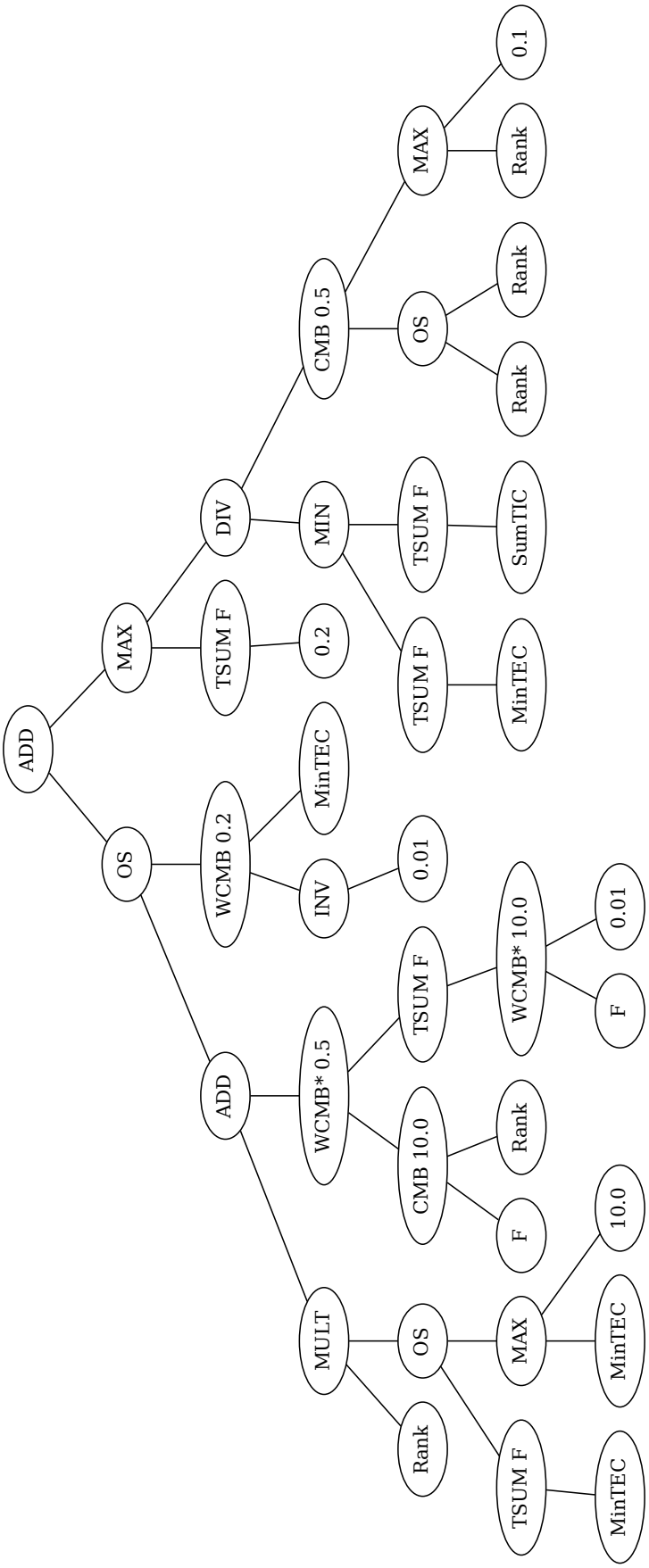


Figura 9.1 – Melhor regra descoberta.
Fonte: O Autor.

9.5.1 Simplificações

Na árvore da melhor regra descoberta, assim como em árvores produzidas pelo algoritmo genético em geral, é possível identificar certas sub-árvores que poderiam ser transformadas em expressões mais simples. A Figura 9.2 apresenta quatro pontos possíveis de simplificação, explicados a seguir. Notamos porém que manter sub-árvores sem ser simplificadas durante a execução do algoritmo genético pode ser vantajoso, visto que existirão mais pontos possíveis para a realização de mutações e recombinações, possibilitando a descoberta de novas árvores.

A: A expressão sempre resulta na constante -0.01.

B: Como a soma do menor tempo excluindo o trabalhador atual sempre será menor ou igual que a soma de todos os tempos incluindo o atual, a expressão poderia ser substituída por (TSUM F (MinTEC)).

C: Não é necessário fazer uma soma sobre todos os sucessores, visto que o resultado sempre será 0.2F.

D: A soma realizada pelo operador OS é convexa, por isso o resultado da expressão sempre será Rank.

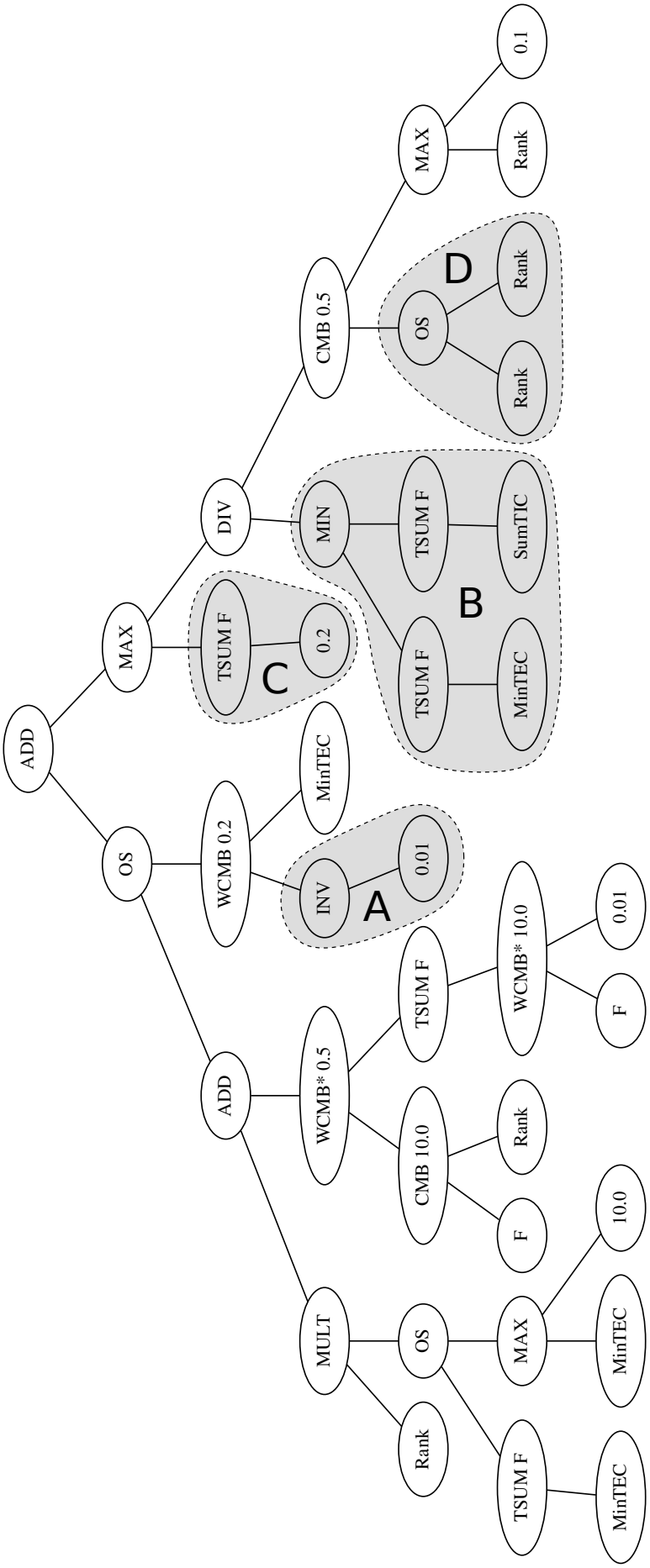


Figura 9.2 – Simplificações possíveis na melhor regra.
Fonte: O Autor.

10 CONCLUSÕES

Neste trabalho apresentamos a aplicação da técnica programação genética ao problema de balanceamento de linhas de produção e designação de trabalhadores (ALWABP), para a descoberta automática de regras de seleção de tarefas usadas por métodos heurísticos para o problema. Explicamos como funciona um procedimento de alocação orientado a estações (SoAP), que é um componente central de heurísticas para o ALWABP, assim como as principais regras descritas na literatura.

Mostramos que as regras de seleção de tarefas podem ser representadas como combinações de elementos simples chamados de *building blocks* e elaboramos uma linguagem capaz de descrever um espaço grande de regras de seleção como diferentes combinações de building blocks. Implementamos um algoritmo genético para buscar de forma eficiente regras de alta qualidade neste espaço e apresentamos estratégias para aprimorar a busca do algoritmo, assim como estratégias lógicas que podem ser aplicadas ao ALWABP de forma geral.

Realizamos experimentos com o nosso algoritmo genético usando a heurística construtiva proposta por MOREIRA et al. (2012) para definir o valor de fitness dos indivíduos, e posteriormente avaliamos os melhores indivíduos encontrados usando a mesma heurística construtiva e também um método que faz parte do estado da arte para o ALWABP, a busca em feixe probabilística iterada (IPBS) proposta por BORBA e RITT (2014).

A Tabela 9.5 mostra que as regras descobertas pelo algoritmo genético consistentemente produzem resultados superiores às melhores regras elaboradas manualmente descritas na literatura. O desvio padrão sobre a qualidade das regras descobertas é baixo, e mesmo o pior resultado sobre todas as replicações para as duas regras avaliadas é superior ao melhor resultado produzido pela melhor regra manual existente (MaxPW^{-}), nas três direções de alocação.

A Tabela 9.6 mostra que quando aplicadas em uma das melhores heurísticas existentes, as regras descobertas também produzem resultados que são competitivos com as melhores regras conhecidas. Mostramos resultados da avaliação com a melhor regra encontrada sobre todos os testes realizados neste trabalho na Tabela 9.7, a qual conseguiu produzir resultados ainda melhores, sustentando a constatação de que as regras descobertas conseguem aprimorar a qualidade de soluções tanto para heurísticas mais simples quanto para o estado da arte.

Os resultados mostram que a programação genética é eficiente em descobrir de forma automática regras de seleção de tarefas para o ALWABP que são superiores às melhores regras já descritas na literatura até a presente data. A chave para o sucesso em programação genética nos parece ser a identificação de elementos simples mas expressivos o suficiente, que possam ser combinados na forma de programas representando soluções para o problema trabalhado. Também é importante a elaboração de estratégias capazes de aprimorar a busca do algoritmo genético por soluções de alta qualidade, a fim de obter da aplicação do método resultados potencialmente melhores.

REFERÊNCIAS

ARAÚJO, F. F. B. **Balanceamento de linhas de produção com trabalhadores deficientes e máquinas paralelas**. Tese (Mestrado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo, São Carlos, maio 2011. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-03062011-142052/>.

BAYBARS, I. A survey of exact algorithms for the simple assembly line balancing problem. **Manage. Sci.**, INFORMS, Linthicum, MD, USA, v. 32, n. 8, p. 900–932, ago. 1986. ISSN 0025-1909.

BLUM, C.; MIRALLES, C. On solving the assembly line worker assignment and balancing problem via beam search. **Computers & Operations Research**, v. 38, n. 2, p. 328–339, 2011.

BORBA, L.; RITT, M. A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. **Computers & Operations Research**, v. 45, p. 87–96, 05 2014.

CHAVES, A.; LORENA, L.; CRISTÓBAL, M. Clustering search approach for the assembly line worker assignment and balancing problem. **37th International Conference on Computers and Industrial Engineering 2007**, v. 2, 01 2007.

CRISTÓBAL, M.; GARCIA-SABATER, J.; ROMANO, C.; CARDÓS, M. Advantages of assembly lines in sheltered work centres for disabled. a case study. **International Journal of Production Economics**, v. 110, p. 187–197, 10 2007.

FUKUNAGA, A. Automated discovery of local search heuristics for satisfiability testing. **Evolutionary Computation**, v. 16, p. 31–61, 2008.

HOFFMANN, T. R. Assembly line balancing: a set of challenging problems. **International Journal of Production Research**, Taylor & Francis, v. 28, n. 10, p. 1807–1815, 1990.

MOREIRA, J. P. G.; RITT, M. Evolving task priority rules for heterogeneous assembly line balancing. In: **2019 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2019. p. 1423–1430.

MOREIRA, M. C.; RITT, M.; COSTA, A. M.; CHAVES, A. A. Simple heuristics for the assembly line worker assignment and balancing problem. **Journal of Heuristics**, Kluwer Academic Publishers, USA, v. 18, n. 3, p. 505–524, Junho 2012. ISSN 1381-1231.

MUTLU, Ö.; POLAT, O.; AYCA, A. An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. **Computers & Operations Research**, v. 40, n. 1, p. 418–426, Janeiro 2013.

OTTO, A.; OTTO, C. How to design effective priority rules: Example of simple assembly line balancing. **Computers & Industrial Engineering**, v. 69, 01 2013.

POLAT, O.; KALAYCI, C. B.; MUTLU, Ö.; GUPTA, S. M. A two-phase variable neighbourhood search algorithm for assembly line worker assignment and balancing problem type-ii: an industrial case study. **International Journal of Production Research**, v. 54, n. 3, p. 722–741, 2016.

POLI, R.; LANGDON, W.; MCPHEE, N. **A Field Guide to Genetic Programming**. [S.l.: s.n.], 2008. ISBN 978-1-4092-0073-4.

SALVESON, M. E. The assembly line balancing problem. **Journal of Industrial Engineering**, v. 6, n. 3, p. 18–25, 1955.

SCHOLL, A. **Balancing and sequencing of assembly lines**. Heidelberg: Physica, 2nd edition Springer, 1999.

SCHOLL, A.; BECKER, C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. **European Journal of Operational Research**, v. 168, n. 3, p. 666–693, 2006.

SCHOLL, A.; VOSS, S. Simple assembly line balancing – heuristic approaches. **Journal of Heuristics**, v. 2, p. 217–244, 1996.