CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

BRUNO FERREIRA ROSA

PROBLEMAS DE PROGRAMAÇÃO DE TAREFAS COM JANELAS DE CONCLUSÃO E PENALIDADES POR ANTECIPAÇÃO E ATRASO: ALGORITMOS E FORMULAÇÕES

BELO HORIZONTE
2017

Centro Federal de Educação Tecnológica de Minas Gerais
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Modelagem Matemática e Computacional

PROBLEMAS DE PROGRAMAÇÃO DE TAREFAS COM JANELAS DE CONCLUSÃO E PENALIDADES POR ANTECIPAÇÃO E ATRASO: ALGORITMOS E FORMULAÇÕES

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, como parte dos requisitos necessários para a obtenção do título de Doutor em Modelagem Matemática e Computacional.

Aluno: Bruno Ferreira Rosa
Orientador: Prof. Dr. Marcone Jamilson Freitas Souza (UFOP, Brasil)
Co-Orientador: Prof. Dr. Philippe Yves Paul Michelon (Université d'Avignon, França)
Co-Orientador: Prof. Dr. Sérgio Ricardo de Souza (CEFET-MG, Brasil)

Belo Horizonte, Outubro de 2017.

Rosa, Bruno Ferreira
R788p Problemas de programação de tarefas com janelas de conclusão e penalidades por antecipação e atraso: algoritmos e formulações. / Bruno Ferreira Rosa. – – Belo Horizonte, 2017.
xvii, 97 f. : il.

Tese (doutorado) – Centro Federal de Educação Tecnológica de Minas Gerais, Programa de Pós-Graduação em Modelagem Matemática e Computacional, 2017.
Orientador: Prof. Dr. Marcone Jamilson Freitas Souza
Coorientador: Prof. Dr. Philippe Yves Paul Michelon
Coorientador: Prof. Dr. Sérgio Ricardo de Souza

Bibliografia

1. Algoritmos Computacionais. 2. Heurística. 3. Programação (Matemática). I. Souza, Marcone Jamilson Freitas. II. Centro Federal de Educação Tecnológica de Minas Gerais. III. Título

CDD 519.7

Elaboração da ficha catalográfica pela Biblioteca-Campus II / CEFET-MG

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

Problemas de Programação de Tarefas em uma Máquina com Janelas de Conclusão e Penalidades por Antecipação e Atraso: Algoritmos e Formulações

Tese de Doutorado apresentada por **Bruno Ferreira Rosa**, em 27 de outubro de 2017, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Prof. Dr. Marcone Jamilson Freitas Souza
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Dr. Sérgio Ricardo de Souza
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Philippe Yves Paul Michelon
Université d'Avignon et des Pays du Vaucluse

Profa. Dra. Débora Pretti Ronconi
Universidade de São Paulo

Prof. Dr. Luiz Satoru Ochi
Universidade Federal Fluminense

Prof. Dr. Maurício Cardoso de Souza
Universidade Federal de Minas Gerais

Prof. Dr. Moacir Felizardo de França Filho
Centro Federal de Educação Tecnológica de Minas Gerais

Profa. Dra. Elisangela Martins de Sá
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Flávio Vinícius Cruzeiro Martins
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida à impressão,

Prof. Dr. José Geraldo Peixoto de Faria
Coordenador do Programa de Pós-Graduação *Stricto Sensu* em Modelagem Matemática e Computacional

Prof. Dr. José Geraldo Peixoto de Faria
Subcoordenador do Programa de Pós-Graduação
em Modelagem Matemática e Computacional
Portaria DIR 325/16, de 06/04/2016

Dedico este trabalho à minha família, meus maiores incentivadores. Amo muito vocês!

Agradecimentos

Agradeço a Deus por minha vida, minha saúde e por me abençoar todos os dias.
Aos meus pais pelo apoio em todos os momentos, sempre torcendo e rezando por mim.
Aos meus irmãos pela amizade e por estarem sempre ao meu lado, principalmente nos momentos de descontração.
À minha esposa, Fernanda, o grande amor da minha vida, por ser minha confidente, minha companheira inseparável, pela compreensão e pelo seu amor incondicional.
Aos professores Marcone e Sérgio por me aceitarem como aluno de doutorado mesmo sabendo das minhas dificuldades e limitações, por acreditarem na minha capacidade, pelas essenciais orientações e apoio, pela paciência e disposição em me ajudar.
Ao professor Philippe Michelon por ter me recebido por um ano na UAPV, ter me orientado e me dado todo o suporte necessário durante esse período.
Aos professores e funcionários do Programa de Pós-Graduação do CEFET-MG por estarem sempre disponíveis para ajudar.
Aos amigos do MMC e da UAPV que estiveram presentes nas etapas deste trabalho. A amizade e colaboração deles foram fundamentais.
Ao CEFET-MG, à UAPV, ao CNPq e à UFOP pelo suporte.
Enfim, os meus sinceros agradecimentos a todos aqueles que de alguma forma colaboraram para a realização deste trabalho.

Resumo

Este trabalho trata o problema de programação de tarefas em uma máquina com janelas de conclusão distintas e tempos de preparação da máquina dependentes da sequência de execução das tarefas, denominado SMSPETP-SDS. O objetivo é minimizar a soma ponderada das antecipações e dos atrasos na conclusão das tarefas. Em termos práticos, as penalidades por antecipação são decorrentes de custos gerados pela necessidade de estocagem, enquanto as penalidades por atraso são consequências de multas contratuais. O SMSPETP-SDS possui muitas aplicações em indústrias metalúrgicas, têxteis, químicas, entre outras. Além do grande número de aplicações, é um problema difícil de ser resolvido na otimalidade, visto pertencer à classe NP-difícil. A união entre a aplicabilidade e a dificuldade de encontrar uma solução ótima motiva o desenvolvimento de algoritmos eficientes para resolvê-lo. Apesar disso, o problema de programação de tarefas com as características consideradas neste trabalho ainda não recebeu a devida atenção. O SMSPETP-SDS tem sido tratado basicamente por meio de procedimentos heurísticos que dividem o problema em dois subproblemas: determinar a melhor programação de uma dada sequência de tarefas, considerando-se a possibilidade de inserção de tempos ociosos entre a execução de tarefas consecutivas; e determinar uma sequência de tarefas que, associada à sua programação ótima, minimize a soma das penalidades geradas pelas tarefas. Neste trabalho, o SMSPETP-SDS é tratado sob uma perspectiva ainda não considerada na literatura. Inicialmente é proposto um novo algoritmo de programação ótima de uma dada sequência de tarefas. Esse algoritmo, de complexidade $O(n^2)$, é utilizado nos algoritmos heurísticos propostos para resolver o problema de sequenciamento das tarefas. Esse algoritmo de programação ótima também é utilizado em um algoritmo exato de enumeração implícita para o caso particular com tempos de preparação da máquina independentes da sequência de execução das tarefas, denominado SMSPETP-SIS. O algoritmo de enumeração implícita proposto faz uso de resultados teóricos desenvolvidos exclusivamente para o SMSPETP-SIS. Em um segundo momento, propõem-se várias formulações matemáticas para o SMSPETP-SDS. Um horizonte de planejamento para a execução de cada tarefa é proposto a fim de ser utilizado na determinação dos parâmetros de entrada dessas formulações. Por último, são propostas novas famílias de restrições válidas para as formulações baseadas em variáveis indexadas no tempo, bem como algoritmos de separação para essas famílias. Experimentos computacionais mostram que: o algoritmo de programação ótima de uma dada sequência de execução das tarefas proposto é mais rápido que o algoritmo até então utilizado para esse fim; os algoritmos heurísticos propostos para o problema de sequenciamento das tarefas são melhores que dois algoritmos da literatura na maioria dos problemas-teste considerados; o algoritmo de enumeração implícita é uma boa alternativa para a resolução exata do SMSPETP-SIS; e os limites inferiores construídos com os algoritmos de separação propostos são muito melhores que as soluções das respectivas relaxações lineares das formulações matemáticas apresentadas.

Palavras-Chave: Programação de Tarefas, Janelas de Conclusão, Preparação da Máquina, Inserção de Tempos Ociosos, Formulações Matemáticas, Heurísticas de Separação.

Abstract

This work addresses the single machine scheduling problem with distinct completion windows and sequence dependent setup times (SMSPETP-SDS). The objective is to minimize the total weighted earliness and tardiness. In the practice, earliness penalties are due to costs generated by the need of stockpiling, while tardiness penalties are consequences of contractual penalties. SMSPETP-SDS has many applications in JIT manufacturing, semi-conductor manufacturing, chemical processing, PERT/CPM scheduling, and video on demand services, among others. In addition, SMSPETP-SDS is an NP-hard problem. The applicability of the SMSPETP-SDS coupled with the difficulty of finding an optimal solution motivates the development of efficient algorithms to it. Nevertheless, the job scheduling problem with the characteristics considered in this work has not received the deserved attention. The SMSPETP-SDS has been treated basically by heuristic procedures that divide the problem into two subproblems: sequencing the jobs, and determining the optimal time for completion of each job in a given sequence (or inserting idle time between jobs in the sequence) in order to minimize the total penalties generated. In this work, the SMSPETP-SDS is treated from a perspective not yet considered in the literature. First, a new $O(n^2)$ optimal scheduling algorithm is proposed. This algorithm is used in proposed heuristic algorithms to solve the job sequencing problem. The optimal scheduling algorithm is also used in an exact algorithm, based on implicit enumeration, for the special case with sequence-independent setup times (SMSPETP-SIS). The proposed implicit enumeration algorithm takes advantage of theoretical results generated for the SMSPETP-SIS. In a second moment, several mathematical formulations are proposed for the SMSPETP-SDS. In order to determine the input parameters of these formulations, a planning horizon for the processing of each job is proposed. Finally, new valid constraints families for the formulations based on time-indexed variables, as well as separation algorithms for these families, are proposed. Computational experiments show that: the new algorithm for the optimal scheduling of a given job sequence is more efficient than the only other equivalent algorithm found in the literature; the proposed heuristic algorithms for the job sequencing problem are better than two algorithms of the literature in most of the instances considered; the implicit enumeration algorithm is a good alternative to the exact resolution of SMSPETP-SIS; and the lower bounds generated with the proposed separation algorithms are much better than the solutions of the respective linear relaxations of the presented mathematical formulations.

Keyword: Job Scheduling, Completion Windows, Setup Times, Idle Time Insertion, Mathematical Formulations, Separation Heuristics.

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Lista de Notações

JIT *Just in Time*

SMSPETP Problema de programação de tarefas em uma máquina com janelas de conclusão distintas e penalidades por antecipação e atraso da conclusão no qual é permitida a ociosidade de máquina entre a execução de tarefas consecutivas – *Single Machine Scheduling Problem with Earliness and Tardiness Penalties*

SMSPETP-SDS SMSPETP com tempos de preparação da máquina dependentes da sequência de execução das tarefas

SMSPETP-SIS SMSPETP com tempos de preparação da máquina independentes da sequência de execução das tarefas

OTA Algoritmo de Wan & Yen (2002) para determinar a programação ótima de uma dada sequência de execução do SMSPETP-SIS – *Optimal Timing Algorithm*

GVNS Metaheurística *General Variable Neighborhood Search*

$\boldsymbol{n}$ Número de tarefas a serem programadas

$\boldsymbol{I}$ Conjunto das n tarefas a serem programadas

$\boldsymbol{P_x}$ Tempo de execução da tarefa x

$\boldsymbol{E_x}$ Início do período desejado para a conclusão da tarefa x

$\boldsymbol{T_x}$ Término do período desejado para a conclusão da tarefa x

$\boldsymbol{\alpha_x}$ Penalidade por unidade de tempo de antecipação da tarefa x

$\boldsymbol{\beta_x}$ Penalidade por unidade de tempo de atraso da tarefa x

$\boldsymbol{C_x}$ Data de conclusão da tarefa x

$\boldsymbol{e_x}$ Antecipação da tarefa x, isto é, $e_x = \max(0,\ E_x - C_x)$

$\boldsymbol{t_x}$ Atraso da tarefa x, isto é, $t_x = \max(0,\ C_x - T_x)$

$\boldsymbol{\pi}$ Uma programação das tarefas de I

$\boldsymbol{C_x^\pi}$ Data de conclusão da tarefa x na programação π

$\boldsymbol{f(\pi)}$ Função de avaliação de uma programação π de I, $f(\pi) = \sum_{x \in I} \alpha_x \cdot \max\left(0, E_x - C_x^\pi\right) + \sum_{x \in I} \beta_x \cdot \max\left(0, C_x^\pi - T_x\right)$

$\boldsymbol{S_{xy}}$ Tempo necessário para preparar a máquina para executar a tarefa y imediatamente após a tarefa x

$\boldsymbol{s_x^\pi}$ Data de início da tarefa x na programação π

$\boldsymbol{S_{max}^I}$ Maior tempo de preparação da máquina possível contido em uma programação de I

TSP *Traveling Salesman Problem*

$\boldsymbol{S_{UB}^I}$ Limite superior para S_{max}^I

$\boldsymbol{DR_I}$ Desvio relativo de S_{UB}^I em relação ao respectivo valor de S_{max}^I

VNS Metaheurística Busca em Vizinhança Variável – *Variable Neighborhood Search*

EI Algoritmo de enumeração implícita para o SMSPETP-SIS

$\boldsymbol{X}$ Sequência do conjunto de tarefas de I, isto é, $X = (x_1, x_2, \ldots, x_n)$

$\boldsymbol{x_i}$ i-ésima tarefa na sequência X

$\boldsymbol{B}$ Subconjunto de tarefas programadas sem ociosidade de máquina entre elas, $B = \{x_u, x_{u+1}, \ldots, x_v\}$, com $1 \leq u \leq v \leq n$

$\boldsymbol{|B|}$ Cardinalidade do bloco B

$\boldsymbol{prim(B)}$ Primeira tarefa do bloco B

$\boldsymbol{ult(B)}$ Última tarefa do bloco B

$\boldsymbol{Custo_B(\varphi)}$ Custo associado às tarefas de B após deslocar esse bloco por φ unidades de tempo para a esquerda

AAOTO Algoritmo de alocação ótima de tempos ociosos

$\boldsymbol{ult_{x_i}}$ Última tarefa do bloco que contém a tarefa x_i

$\boldsymbol{CM(B)}$ Custo marginal para adiar as tarefas de B por uma unidade de tempo

OTA' Algoritmo OTA melhorado

$\boldsymbol{prim_{x_i}}$ Primeira tarefa do bloco que contém a tarefa x_i

VND Descida em Vizinhança Variável – *Variable Neighborhood Descent*

GVNSmax Parâmetro do GVNS associado ao critério de parada adotado

$\boldsymbol{F(X)}$ Função de avaliação de uma sequência de tarefas X

$\mathbf{VND_r}$ VND que faz uso de descidas randômicas

VNDmax Parâmetro do VND_r associado ao critério de parada adotado

$\mathbf{VND_f}$ VND que faz uso do método de descida com estratégia *First Improvement*

$\mathbf{GVNS_r}$ GVNS que utiliza o VND_r como busca local

$\mathbf{GVNS_f}$ GVNS que utiliza o VND_f como busca local

$\mathbf{GVNS_{rf}}$ Algoritmo obtido ao se aplicar a busca local VND_f na solução obtida com o GVNS_r

$\boldsymbol{UB}$ Limite superior dado para o SMSPETP-SIS

$\boldsymbol{\delta}$ Nó da árvore de busca do algoritmo EI

$\boldsymbol{seq(\delta)}$ Subsequência de k tarefas armazenada no nó δ, $0 \leq k \leq n$

$\boldsymbol{nseq(\delta)}$ Conjunto das $n - k$ tarefas que não estão em $seq(\delta)$

$\boldsymbol{L}$ Lista de nós a serem investigados no algoritmo EI

$\boldsymbol{A}$ Uma subsequência das tarefas de I

$\boldsymbol{X_A}$ Sequência das tarefas em I que inicia com a subsequência A

$\boldsymbol{\pi_A}$ Uma programação da subsequência de tarefas A

$\mathbf{DRM_i^{avg}}$ Métrica desvio relativo médio para um dado problema i

GVNS$_f$/OTA GVNS$_f$ que utiliza o OTA para avaliar uma dada sequência de tarefas

GVNS$_f$/AAOTO GVNS$_f$ que utiliza o AAOTO para avaliar uma dada sequência de tarefas

GVNS$_f$/OTA' GVNS$_f$ que utiliza o OTA' para avaliar uma dada sequência de tarefas

VP Formulação baseada em variáveis de precedência para o SMSPETP-SDS

VP' Formulação VP acrescida de uma família de restrições válidas

VPI Formulação baseada em variáveis de precedência imediata para o SMSPETP-SDS

VPIA Formulação VPI alternativa

$\boldsymbol{s_x^{UB}}$ Limite superior para a data de início da tarefa x

$\boldsymbol{s_x^{LB}}$ Limite inferior para a data de início da tarefa x

VPS Formulação baseada em variáveis de posição na sequência para o SMSPETP-SDS

$\boldsymbol{H_x}$ Conjunto das possíveis datas de início de execução da tarefa x, $H_x = \{s_x^{LB}, s_x^{LB} + 1, \ldots, s_x^{UB}\}$

$\boldsymbol{\lfloor \lambda \rfloor_x}$ $\max(s_x^{LB}, \lambda)$, para toda tarefa $x \in I$ e para todo número real λ

$\boldsymbol{\lceil \lambda \rceil_x}$ $\min(\lambda, s_x^{UB})$, para toda tarefa $x \in I$ e para todo número real λ

IMR Formulação linear inteira mista para o SMSPETP-SDS de Rosa & Souza (2009)

IMN Formulação linear inteira mista para o SMSPETP-SDS baseada em Nogueira *et al.* (2014)

BR Formulação binária para o SMSPETP-SDS obtida da Formulação IMR

BN Formulação binária para o SMSPETP-SDS obtida da Formulação IMN

B1 Primeira formulação binária proposta para o SMSPETP-SDS

B2 Segunda formulação binária proposta para o SMSPETP-SDS

$\boldsymbol{PPM}$ Problema de programação matemática baseado em variáveis indexadas no tempo que é atualizado iterativamente em um dado algoritmo de planos-de-corte

$\boldsymbol{l^*}$ Solução ótima do PPM corrente

$\boldsymbol{h_x^{min}}$ $\min\{h \in H_x : \ l_{xh}^* > 0\}, \ \forall x \in I$

$\boldsymbol{h_x^{max}}$ $\max\{h \in H_x : \ l_{xh}^* > 0\}, \ \forall x \in I$

$\boldsymbol{I_x}$ $\left\{y \in I \setminus \{x\} : h_y^{max} \ + \ P_y \ + \ S_{yx} \ > \ h_x^{min} \ \text{ ou } h_x^{max} + P_x + S_{xy} > h_y^{min}\right\}$

$\boldsymbol{\delta}$ Violação mínima considerada

$\boldsymbol{TPT_{min}^{I'}}$ Menor tempo total necessário para executar todas as tarefas de $I' \subset I$

gap Desvio de um dado limite inferior para o problema em relação a uma solução do respectivo problema

Capítulo 1

Introdução

O surgimento do sistema administrativo *Just in Time* (JIT) evidenciou a importância de um planejamento criterioso das atividades produtivas. Reduzir as antecipações e os atrasos nas execuções das tarefas de uma atividade produtiva pode proporcionar uma significativa redução de custos. De acordo com Baker & Scudder (1990), concluir uma tarefa com atraso, ou seja, após a data desejada para sua conclusão, pode resultar em multas contratuais, perda de credibilidade da empresa e redução de vendas. Do mesmo modo, concluir uma tarefa antecipadamente, isto é, antes da data desejada para sua conclusão, pode resultar em custos financeiros extras pela necessidade de disponibilização antecipada de capital, necessidade de espaço para armazenamento ou necessidade de outros recursos para manter e gerenciar o estoque (Józefowska, 2007).

O problema de programação de tarefas em uma máquina com penalidades por antecipação e atraso consiste em programar um conjunto de tarefas em uma máquina de modo a minimizar a soma ponderada das antecipações e atrasos na conclusão dessas tarefas.

As datas desejadas para a conclusão das tarefas podem ser comuns ou distintas. No entanto, segundo Wan & Yen (2002), devido às incertezas e tolerâncias, em muitas situações da indústria de manufatura é esperado que as tarefas sejam concluídas dentro de determinados intervalos de tempo (janelas de conclusão), em vez de em datas específicas (datas de conclusão). Tais incertezas e tolerâncias estão relacionadas às características individuais das tarefas e influenciam no tamanho das janelas de conclusão. Assim, apenas as tarefas concluídas antes ou após suas respectivas janelas de conclusão estarão sujeitas às penalidades.

Existe uma variedade de aplicações de modelos de programação de tarefas com janelas de conclusão em processos de produção baseados no sistema JIT, na fabricação de materiais semicondutores, em processos químicos, em programações PERT/CPM, em serviços de vídeos sob demanda, dentre outros (Janiak *et al.*, 2015). A produção de bens perecíveis apresentada em Koulamas (1996) é um exemplo dessas aplicações. Suponha que um fabricante de produtos químicos combina um determinado produto químico "A", que se deteriora rapidamente, com um segundo produto químico "B" para produzir um produto "C". Se "A" for produzido antes que "B" esteja pronto, então ele se deteriorará. Por outro lado, se "A" for produzido com atraso, então os atrasos incorridos na produção de "C" podem gerar custos.

O problema de programação de tarefas em uma máquina com penalidades por antecipação e atraso com datas de conclusão comuns é NP-difícil Baker & Scudder (1990), ou seja, ainda não é conhecido nenhum algoritmo de complexidade polinomial que o resolva. Como o caso de datas de conclusão comuns pode ser visto como um caso particular do problema com dadas de conclusão distintas, e consequentemente do problema com janelas de conclusão distintas, tem-se que o problema de programação em uma máquina com penalidades por antecipação e atraso das tarefas com janelas de conclusão também é NP-difícil.

Nas indústrias em que são fabricados diferentes tipos de produtos e existe uma troca fre-

quente do tipo de tarefa executada em uma máquina, geralmente é necessário preparar a máquina entre a execução de tarefas consecutivas (Allahverdi *et al.*, 1999). Esse tempo de preparação inclui os tempos gastos para trocar as ferramentas, preparar o material, limpar a máquina, etc. A maioria dos trabalhos em problemas de programação de tarefas assume que os tempos de preparação da máquina são independentes da sequência de execução das tarefas, isto é, eles são desprezíveis ou acrescentados aos tempos de execução das tarefas (Gupta & Smith, 2006). No entanto, de acordo com Kopanos *et al.* (2009) e Gupta & Smith (2006), os tempos de preparação da máquina são dependentes da sequência de execução das tarefas em grande parte das situações práticas. O caso real de uma indústria siderúrgica, tratado em Bustamante (2007), é ilustrado na Figura 1.1. Nesse trabalho, uma sequência de laminadores é considerada como uma máquina e cada tarefa representa a produção de um determinado produto (barra chata, cantoneira, vergalhão, etc.). Antes da fabricação de cada produto, é necessário realizar um conjunto de ajustes de mesma natureza na sequência de laminadores. Esses ajustes dependem do produto a ser fabricado e do produto fabricado anteriormente. Como cada ajuste exige um tempo para ser realizado, a soma desses tempos configura o tempo de preparação da máquina. Em problemas como esse, os tempos de preparação da máquina variam de acordo com a sequência de produção e representam uma parcela de tempo considerável em relação ao tempo total de execução. Portanto, eles não podem ser desconsiderados. Allahverdi *et al.* (1999), Allahverdi *et al.* (2008) e Allahverdi (2015) fazem uma ampla revisão da literatura a respeito de problemas de programação de tarefas com tempos de preparação da máquina.

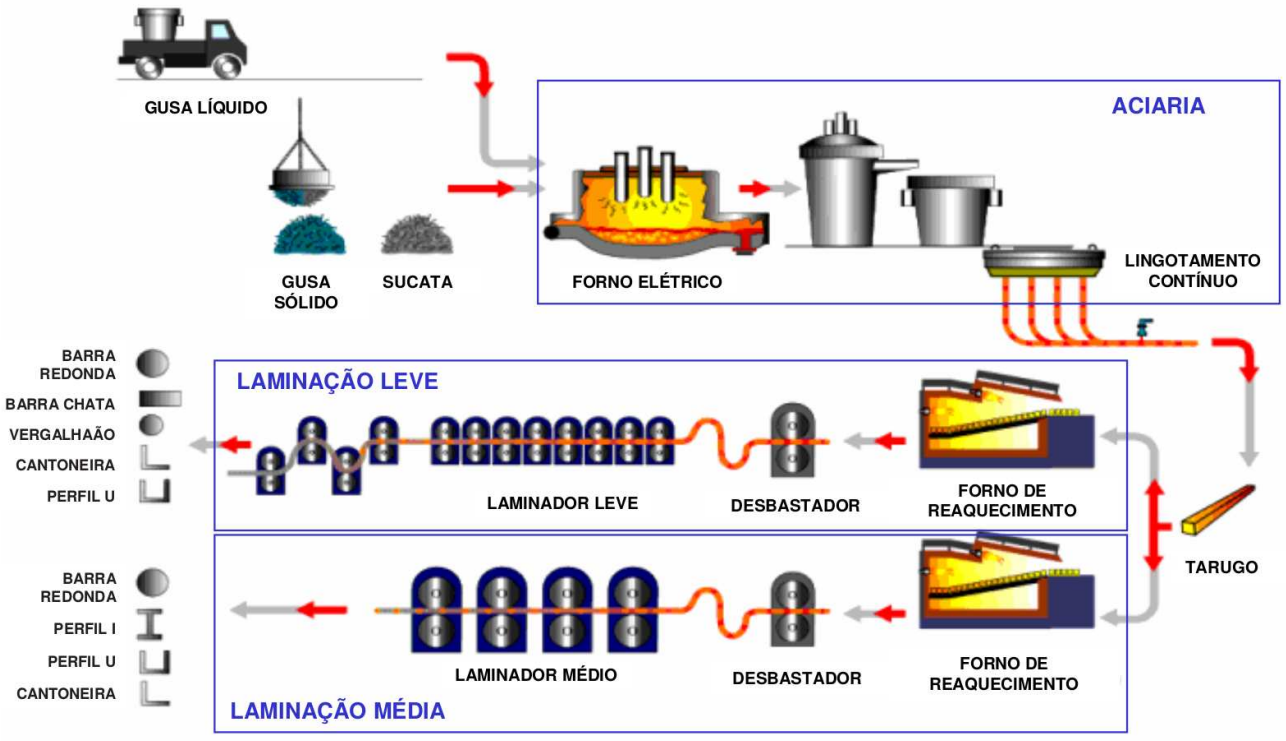


Figura 1.1: Aplicação do problema de programação de tarefas em uma indústria siderúrgica.

Considerações a respeito da continuidade do funcionamento da máquina também podem ocorrer. Conforme Kanet & Sridharan (2000), existem situações em que a ociosidade da máquina não é permitida, por implicar em custos mais elevados que aqueles decorrentes da conclusão antecipada das tarefas. No entanto, o mesmo autor afirma que há casos em que manter a máquina inativa é vantajoso, ainda que exista uma tarefa disponível para ser executada. Assim, não havendo restrições à ociosidade da máquina, determinar a melhor data para iniciar a execução de cada tarefa ou, equivalentemente, permitir ociosidade de máquina entre a execução de tarefas, pode produzir melhores soluções.

Esta tese de doutorado trata o problema de programação de tarefas em uma máquina com janelas de conclusão distintas e penalidades por antecipação e atraso da conclusão, sendo permitida a ociosidade de máquina entre a execução de tarefas consecutivas. No restante deste trabalho, o problema de programação de tarefas com essas características será denotado por SMSPETP (das iniciais em inglês de *Single Machine Scheduling Problem with Earliness and Tardiness Penalties*). O SMSPETP com tempos de preparação da máquina dependentes da sequência de execução das tarefas será doravante denotado por SMSPETP-SDS, sendo o sufixo SDS relativo às iniciais, em inglês, de *Sequence-Dependent Setup*. De acordo com a notação utilizada por Pinedo (2012), o SMSPETP-SDS é representado por $1/s_{jk}/\sum_{j=1}^{n} w'_j E_j + \sum_{j=1}^{n} w''_j T_j$. Por outro lado, o SMSPETP com tempos de preparação da máquina independentes da sequência de execução das tarefas será denotado de agora em diante por SMSPETP-SIS. Pinedo (2012) representa o SMSPETP-SIS por $1/\ /\sum_{j=1}^{n} w'_j E_j + \sum_{j=1}^{n} w''_j T_j$.

O restante deste Capítulo está organizado da seguinte forma. Na Seção 1.1 é feita uma revisão da literatura relacionada ao SMSPETP. Na Seção 1.2 é apresentada a motivação deste trabalho. Na Seção 1.3 são expostos os objetivos geral e específicos traçados, enquanto a Seção 1.4 apresenta a estrutura desta tese.

1.1 Estado da arte

A união entre a aplicabilidade e a dificuldade de encontrar uma solução ótima para o SMSPETP tem motivado diversos trabalhos. Nesta Seção são apresentados trabalhos relacionados ao problema de programação de tarefas objeto desta tese de doutorado.

Koulamas (1996) aborda o SMSPETP-SIS com uma sensível distinção dos demais trabalhos da literatura. Ao contrário dos outros trabalhos, que consideram a existência de antecipação quando uma tarefa é concluída antes do início de sua janela de conclusão, esse autor considera que há antecipação de uma tarefa quando sua execução é iniciada antes do início de tal janela. Além disso, as penalidades por unidade de tempo de antecipação e atraso são consideradas unitárias. O autor divide o problema em dois subproblemas, sendo um deles determinar a sequência de execução das tarefas e o outro programar a data ótima de conclusão de cada tarefa numa dada sequência de execução (ou, equivalentemente, alocar tempos ociosos entre as tarefas de uma dada sequência de execução). Para resolver o problema de programação das datas ótimas de execução de cada tarefa numa dada sequência, é desenvolvido um algoritmo que aloca tempos ociosos, de modo ótimo, na sequência de execução. Já o problema de sequenciamento das tarefas é resolvido com adaptações de heurísticas utilizadas anteriormente em casos particulares do SMSPETP.

Wan & Yen (2002) tratam o SMSPETP-SIS. São apresentadas várias propriedades do problema a fim de facilitar sua resolução e, assim como Koulamas (1996), os autores dividem o problema em dois subproblemas. É desenvolvido um algoritmo de complexidade polinomial para determinar a data ótima de conclusão de cada tarefa em uma dada sequência de execução. Esse algoritmo, de agora em diante denotado por OTA (*Optimal Timing Algorithm*), é uma extensão dos algoritmos de Davis & Kanet (1993), Lee & Choi (1995) e Szwarc & Mukhopadhyay (1995) para o problema de programação de tarefas com datas de conclusão distintas. Por fim, uma Busca Tabu (Glover & Laguna, 1997), que faz uso do procedimento de datas ótimas e das propriedades apresentadas, é utilizada para resolver o problema.

Bustamante (2007) apresenta dois modelos de programação linear inteira mista para o problema de programação de tarefas com datas de conclusão distintas, tempos de preparação da máquina dependentes da sequência de execução das tarefas e penalidades por antecipação e atraso. O primeiro modelo é baseado nas definições de Manne (1960), enquanto o segundo é baseado em Wagner (1959) e não abrange os problemas em que as penalidades por unidade de tempo de antecipação e atraso são dependentes das tarefas. Os modelos apresentados exigem

que os tempos de preparação da máquina satisfaçam a desigualdade triangular (ver Seção 2.1.1). O autor também sugere alterações nos modelos que permitem resolver o problema de programação de tarefas com janelas de conclusão distintas. No entanto, os testes computacionais se concentram em problemas-teste com datas de conclusão distintas, sendo utilizado somente um problema teste com janelas de conclusão distintas. O software de otimização GLPK 4.8 é utilizado para encontrar a solução ótima de problemas-teste com até 10 tarefas.

Gomes Júnior *et al.* (2007) propõem um modelo matemático inteiro misto, baseado em uma das formulações de programação matemática de Bustamante (2007). Porém, esse modelo não exige que os tempos de preparação da máquina satisfaçam à desigualdade triangular. O otimizador CPLEX, versão 9.1, é utilizado para encontrar a solução ótima de problemas-teste com até 12 tarefas e com tempos de preparação da máquina simétricos. Os autores propõem, também, um algoritmo heurístico, baseado em GRASP (Feo & Resende, 1995), *Iterated Local Search* (Lourenço *et al.*, 2003) e *Variable Neighborhood Descent* (Mladenović & Hansen, 1997) para resolvê-lo. Para cada sequência de tarefas gerada pela heurística desenvolvida, aplicou-se o OTA (Wan & Yen, 2002) – adaptado para incluir tempos de preparação da máquina – para programar as datas ótimas de início de execução das tarefas nessa sequência.

O trabalho de Rosa & Souza (2009) apresenta um modelo de programação matemática inteiro misto e indexado no tempo para representar o SMSPETP-SDS. A estimativa do horizonte de planejamento, a qual faz parte dos dados de entrada do modelo, é obtida pela aplicação de um algoritmo heurístico baseado nos procedimentos GRASP, Princípio da Otimalidade Próxima (Glover & Laguna, 1997) e *Variable Neighborhood Descent*. Assim como os modelos de Bustamante (2007), a formulação indexada no tempo necessita que os tempos de preparação da máquina satisfaçam a desigualdade triangular.

São encontrados na literatura muitos trabalhos que fazem uso do OTA adaptado por Gomes Júnior *et al.* (2007) em procedimentos heurísticos para resolver o SMSPETP-SDS. Entre eles, citamos: Souza *et al.* (2008), Rosa *et al.* (2010), Ribeiro *et al.* (2010) e Penna *et al.* (2012).

Nogueira *et al.* (2014) propõem e analisam seis formulações de programação linear inteira mista para o problema de programação de tarefas com tempos de liberação das tarefas distintos e tempos de preparação da máquina dependentes da sequência de execução das tarefas. A fim de testar as formulações propostas, foram consideradas duas funções objetivos: soma ponderada da data de conclusão das tarefas e soma ponderada dos atrasos das tarefas.

Parte dos resultados obtidos durante o desenvolvimento desta tese de doutorado foi publicada em Rosa *et al.* (2017). Nesse trabalho são propostos um algoritmo exato de enumeração implícita para o SMSPETP-SIS e um algoritmo heurístico baseado no método GVNS – *General Variable Neighborhood Search* (Hansen *et al.*, 2008) – para o SMSPETP-SDS. Um algoritmo de complexidade $O(n^2)$ que aloca, de modo ótimo, tempos ociosos em uma dada sequência de execução das tarefas também é proposto. Tanto o algoritmo de enumeração implícita quanto o algoritmo GVNS fazem uso desse algoritmo de alocação ótima de tempos ociosos proposto para avaliar cada sequência de execução construída. Embora o algoritmo de enumeração implícita seja aplicável apenas ao SMSPETP-SIS, ele faz uso de resultados teóricos gerados para essa classe do SMSPETP.

1.2 Motivação

De acordo com o que foi apresentado anteriormente, o SMSPETP consiste em uma generalização do problema de programação de tarefas e possui muitas aplicações práticas. No entanto, essa versão do problema ainda não recebeu a atenção merecida. Até então, seu tratamento tem sido somente por meio de procedimentos heurísticos. Apesar de Bustamante (2007), Gomes Júnior *et al.* (2007) e Rosa & Souza (2009) apresentarem formulações de programação matemática

que representam o problema, as formulações propostas foram utilizadas apenas em *softwares* genéricos e que não exploram as características particulares do SMSPETP na sua resolução.

Resolver o SMSPETP consiste em determinar a sequência na qual as tarefas serão executadas, bem como programar a data de início de execução de cada tarefa nessa sequência. Os trabalhos encontrados na literatura tratam o problema com procedimentos heurísticos que, iterativamente, geram uma sequência de execução das tarefas por meio de regras de prioridade ou procedimentos de busca. Para avaliar cada sequência gerada é necessário programar a data ótima de início de execução de cada tarefa em tal sequência, permitindo ociosidade de máquina entre a execução de tarefas consecutivas. Sendo assim, é fundamental o uso de um algoritmo que faça a alocação de tempos ociosos o mais rápido possível. Todos os trabalhos encontrados na literatura fazem uso do algoritmo OTA adaptado por Gomes Júnior *et al.* (2007) para avaliar uma dada sequência de execução de tarefas do SMSPETP, visto que trata-se do único algoritmo encontrado para esse fim. Procurou-se, então, desenvolver um algoritmo mais rápido que o OTA para programar as datas ótimas de início de execução das tarefas do SMSPETP.

A presente tese de doutorado aborda o SMSPETP de uma forma ainda não considerada na literatura. O problema é tratado tanto com métodos exatos quanto com métodos heurísticos. A principal finalidade em se adotar abordagens exatas é determinar bons limites inferiores para o problema, ao passo que as abordagens heurísticas visam proporcionar bons limites superiores.

1.3 Objetivos

Nesta seção são apresentados os objetivos deste trabalho. O objetivo geral é apresentado na Seção 1.3.1, enquanto os objetivos específicos são apresentados na Seção 1.3.2.

1.3.1 Objetivo geral

O principal objetivo da presente tese de doutorado é desenvolver métodos exatos e heurísticos, bem como métodos híbridos, que utilizam as particularidades do SMSPETP para resolvê-lo de forma eficiente. Os métodos desenvolvidos deverão ser capazes de competir com os algoritmos atualmente existentes na literatura relacionada ao problema, bem como servir de referência para futuras pesquisas que envolvam problemas de programação de tarefas com janelas de entrega e tempos de preparação da máquina dependentes da sequência de execução das tarefas.

1.3.2 Objetivos específicos

Com a finalidade de alcançar o objetivo geral, os seguintes objetivos específicos foram traçados:

- Fazer um levantamento de trabalhos da literatura que tratam o problema abordado ou problemas relacionados, bem como dos métodos utilizados para resolvê-los;
- Determinar propriedades matemáticas particulares do SMSPETP que possam auxiliar tanto os métodos exatos quanto os métodos heurísticos a resolvê-lo;
- Desenvolver um novo algoritmo de programação ótima de uma dada sequência de execução de tarefas do SMSPETP;
- Desenvolver um novo algoritmo heurístico para o SMSPETP;
- Desenvolver um algoritmo de enumeração implícita para o SMSPETP;

- Desenvolver novas formulações de programação matemática para o problema;
- Desenvolver novas restrições válidas (cortes) a fim de fortalecer as formulações desenvolvidas;
- Desenvolver um algoritmo de inserção de cortes para as formulações de programação matemática desenvolvidas;
- Realizar testes computacionais usando os métodos propostos;
- Analisar os resultados obtidos e compará-los com os de outros métodos da literatura;
- Divulgar os resultados obtidos em eventos científicos e em periódicos especializados.

1.4 Estrutura do trabalho

Este trabalho está organizado como segue.

No Capítulo 1 é feita uma introdução ao problema estudado. São expostos os objetivos traçados, uma revisão da literatura relacionada ao problema em foco e também a motivação deste trabalho.

No Capítulo 2 é feita uma descrição detalhada do SMSPETP, bem como das versões desse problema que são tratadas neste trabalho. Também é feita uma apresentação dos problemas-teste utilizados nos experimentos computacionais com os métodos propostos. Finalmente, propõe-se resultados teóricos que permitem calcular um horizonte de planejamento para a execução de cada tarefa do SMSPETP.

No Capítulo 3 são propostos algoritmos heurísticos a fim de prover limites superiores para o SMSPETP. Três algoritmos de programação ótima de uma dada sequência de tarefas também são apresentados. Um deles é um algoritmo da literatura, enquanto os outros dois são contribuições deste trabalho. A metaheurística Busca em Vizinhança Variável (*Variable Neighborhood Search*) é descrita e algoritmos baseados nessa metaheurística para resolver o SMSPETP-SDS são propostos. Além disso, propõe-se um algoritmo de enumeração implícita para o SMSPETP-SIS. Por fim, apresentam-se e analisam-se os resultados computacionais obtidos com os métodos desenvolvidos e conclui-se o capítulo.

No Capítulo 4 são apresentadas formulações matemáticas que representam o SMSPETP-SDS. Entre elas, estão formulações baseadas em variáveis de precedência, formulações baseadas em variáveis de precedência imediata, uma formulação baseada em variáveis de posição na sequência e formulações baseadas em variáveis indexadas no tempo. São propostas cinco novas famílias de restrições válidas para as formulações baseadas em variáveis indexadas no tempo, bem como algoritmos de separação para essas famílias de restrições. Por último, apresentam-se e discutem-se os resultados computacionais obtidos com os métodos apresentados e conclui-se o capítulo.

O Capítulo 5 apresenta as conclusões finais desta tese de doutorado e, também, sugestões de trabalhos futuros.

Capítulo 2

Problema abordado

Neste capítulo é feita a formalização do problema tratado neste trabalho. O SMSPETP é descrito detalhadamente na Seção 2.1. Na Seção 2.2 são apresentados os problemas-teste utilizados na avaliação dos métodos propostos, enquanto resultados teóricos que possibilitam estabelecer um horizonte de planejamento para a execução de cada tarefa do SMSPETP são propostos na Seção 2.3.

2.1 Características do SMSPETP

O problema de programação de tarefas tratado neste trabalho (SMSPETP – *Single Machine Scheduling Problem with Earliness and Tardiness Penalties*) possui as seguintes características:

- Uma máquina deve executar um conjunto I de n tarefas;
- Cada tarefa $x \in I$ possui:
  - um tempo de execução P_x;
  - uma janela de conclusão $[E_x, T_x]$, dentro da qual a tarefa x deve ser preferencialmente concluída. E_x e T_x correspondem ao início e ao fim do período desejado para a conclusão da tarefa x, respectivamente;
  - uma penalidade α_x por unidade de tempo de antecipação ;
  - uma penalidade β_x por unidade de tempo de atraso;
- Há antecipação de uma tarefa $x \in I$ se, e somente se, sua execução for concluída antes de E_x. Portanto, a antecipação de x é dada por $e_x = \max(0,\ E_x - C_x)$, em que C_x representa a data de conclusão de x;
- Há atraso de uma tarefa $x \in I$ se, e somente se, sua execução for concluída após T_x. Portanto, o atraso de x é dado por $t_x = \max(0,\ C_x - T_x)$, em que C_x representa a data de conclusão de x;
- As tarefas concluídas dentro de suas respectivas janelas de conclusão não geram penalidades;
- A máquina pode executar no máximo uma tarefa por vez;
- Uma vez iniciada a execução de uma tarefa, não é permitida a sua interrupção;
- Todas as tarefas estão disponíveis para serem executadas a partir da data 0;

- É permitida a ociosidade de máquina, ou seja, a máquina pode ficar ociosa entre a execução de duas tarefas consecutivas.

O objetivo é determinar uma programação de I (isto é, uma alocação de tempo para cada tarefa $x \in I$ em uma dada máquina) que minimize o custo oriundo de antecipações e de atrasos. Em outras palavras, o objetivo do SMSPETP é encontrar uma programação π de I que minimize a função:

$$f(\pi) = \sum_{x \in I} \alpha_x \cdot \max\left(0,\ E_x - C_x^\pi\right) + \sum_{x \in I} \beta_x \cdot \max\left(0, C_x^\pi - T_x\right), \tag{2.1}$$

em que C_x^π representa a data de conclusão da tarefa x na programação π.

A seguir são detalhadas as classes do SMSPETP abordadas neste trabalho. Na Seção 2.1.1 é descrito o SMSPETP com tempos de preparação da máquina dependentes da sequência de execução das tarefas, denotado por SMSPETP-SDS. Já na Seção 2.1.1 é descrito o SMSPETP com tempos de preparação da máquina independentes da sequência de execução das tarefas, representado por SMSPETP-SIS.

2.1.1 SMSPETP-SDS

No SMSPETP com tempos de preparação da máquina dependentes da sequência de execução das tarefas (SMSPETP-SDS) entre a execução de duas tarefas consecutivas x e $y \in I$ é necessário um tempo de preparação da máquina S_{xy}. Assume-se que o tempo necessário para preparar a máquina para a execução da primeira tarefa programada é igual a 0. Além disso, assume-se que os tempos de preparação da máquina satisfazem a desigualdade triangular, isto é, as inequações (2.2) são satisfeitas:

$$S_{xz} \ \leq \ S_{xy} + P_y + S_{yz}, \qquad \forall\, x,\, y,\, z \in I,\ x \neq y,\ x \neq z \ \text{ e } \ y \neq z. \tag{2.2}$$

As inequações (2.2) asseguram que não é vantajoso efetuar duas preparações da máquina (executando uma terceira tarefa entre as tarefas envolvidas) em vez de uma. Por exemplo, não pode ser vantajoso fazer a preparação da máquina da tarefa x para a tarefa y, executar a tarefa y e preparar a máquina para a tarefa z, em vez de fazer a preparação da máquina da tarefa x para a tarefa z. A Figura (2.1) ilustra essa situação, que é uma condição necessária para que algumas formulações de programação matemática representem o SMSPETP-SDS.

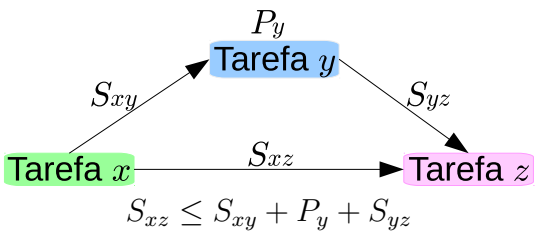


Figura 2.1: Desigualdade triangular.

Observa-se que a desigualdade triangular considerada neste trabalho é menos restritiva que a definição de desigualdade triangular que envolve apenas os tempos de preparação da máquina, dada pelas inequações (2.3).

$$S_{xz} \ \leq \ S_{xy} + S_{yz}, \qquad \forall\, x,\, y,\, z \in I,\ x \neq y,\ x \neq z \ \text{ e } \ y \neq z. \tag{2.3}$$

2.1.2 SMSPETP-SIS

O caso particular do SMSPETP com tempos de preparação da máquina independentes da sequência de execução das tarefas (SMSPETP-SIS) também é considerado neste trabalho. Neste caso, o valor de S_{xy} é o mesmo para qualquer par de tarefas x, $y \in I$ e, por isso, pode ser desconsiderado ou adicionado ao tempo de execução das tarefas.

2.2 Descrição dos problemas-teste

A fim de avaliar os métodos desenvolvidos neste trabalho para resolver o SMSPETP, foram gerados problemas-teste com base nos procedimentos descritos em Wan & Yen (2002) e Rabadi *et al.* (2004). O conjunto de problemas-teste gerado é descrito a seguir.

Dada uma tarefa $x \in I$, o tempo de execução (P_x), o custo por unidade de atraso (β_x) e o custo por unidade de antecipação (α_x) são números inteiros selecionados aleatoriamente, usando uma distribuição uniforme, dentro dos intervalos $[1, 40]$, $[1, 10]$ e $[1, \beta_x]$, respectivamente. O centro da janela de conclusão da tarefa x é um número inteiro, selecionado de forma aleatória e com distribuição uniforme, dentro do intervalo

$$\left[\left(1 - FA - \frac{VRJ}{2}\right) \times TTP, \left(1 - FA + \frac{VRJ}{2}\right) \times TTP\right],$$

em que:

- TTP é o tempo total de execução de todas as tarefas;
- FA é o fator de atraso; e
- VRJ é a variação relativa da janela de conclusão.

O tamanho da janela de conclusão é um número inteiro selecionado aleatoriamente, usando uma distribuição uniforme, no intervalo $\left[0, \frac{TTP}{n}\right]$, no qual n representa o número de tarefas a serem executadas. Para cada tarefa $y \in I$ e $y \neq x$, o tempo de preparação da máquina S_{xy} é um número inteiro escolhido aleatoriamente, usando uma distribuição uniforme, no intervalo $[5, 15]$. Os tempos de preparação da máquina não são necessariamente simétricos, isto é, S_{xy} e S_{yx} foram escolhidos de forma independente. Foram gerados conjuntos de problemas-teste com 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 50, 75 e 100 tarefas, respectivamente, sendo utilizados os valores 0,1; 0,3; 0,5 e 0,8 para FA e 0,4; 0,7; 1,0 e 1,3 para VRJ. Portanto, há 16 problemas-teste em cada conjunto e um total de 320 problemas-teste. Toda vez que um problema-teste gerado não satisfazia a desigualdade triangular, dada pelas inequações (2.2), o problema era descartado e um outro problema-teste com os mesmos valores de FA e VRJ era gerado. Logo, todos os problemas-teste da base de dados utilizada satisfazem a desigualdade triangular em relação aos tempos de preparação da máquina.

Os problemas-teste utilizados neste trabalho estão disponíveis em `http://www.decom.ufop.br/prof/marcone/projects/SMSPETP.html`.

2.3 Horizonte de planejamento

Uma programação ótima para o SMSPETP não é necessariamente aquela de menor *makespan*, isto é, uma programação de menor data de conclusão da última tarefa executada (Pinedo, 2012; Nogueira *et al.*, 2014). Embora o horizonte de planejamento para a execução das tarefas do SMSPETP-SDS seja muito importante na determinação dos parâmetros de entrada das formulações matemáticas que representam esse problema (para as formulações baseadas

em variáveis indexadas no tempo, por exemplo, o horizonte de planejamento das tarefas influencia diretamente no número de variáveis do modelo), não foi encontrado, até então, nenhum trabalho que faça um estudo do horizonte de planejamento para as tarefas do SMSPETP-SDS.

Esta seção tem o objetivo de estabelecer um horizonte de planejamento para a execução das tarefas de I. O horizonte de planejamento estabelecido deve garantir que a solução ótima do problema esteja contida nele.

Dadas uma programação π de I e uma tarefa $x \in I$, s_x^π representa a data de início da tarefa x em π e C_x^π representa a data de conclusão da tarefa x em π.

Proposição 2.1 *Seja π^* uma programação ótima e π uma possível programação de I. Então $\max_{x\in I} C_x^{\pi^*}$ não é necessariamente menor que $\max_{x\in I} C_x^\pi$.*

Prova: A prova será realizada por meio de um contra-exemplo. Considere o SMSPETP-SIS com duas tarefas tais que:

- a tarefa 1 possui o tempo de execução $P_1 = 4$, a janela de conclusão $[4, 6]$ e as penalidades por antecipação e atraso $\alpha_1 = 1$ e $\beta_1 = 2$, respectivamente;
- a tarefa 2 possui o tempo de execução $P_2 = 3$, a janela de conclusão $[4, 5]$ e as penalidades por antecipação e atraso $\alpha_2 = 3$ e $\beta_2 = 4$, respectivamente;

A Figura 2.2 mostra uma possível programação π de $I = \{1, 2\}$ em que $\max_{x\in I} C_x^\pi = 7$ e $f(\pi) = 2 \times \beta_2 = 8$. Por outro lado, a Figura 2.3 mostra a programação ótima π^* de $I = \{1, 2\}$, a qual possui $f(\pi^*) = 2 \times \beta_1 = 2$ e $\max_{x\in I} C_x^{\pi^*} = 8 > \max_{x\in I} C_x^\pi$. □

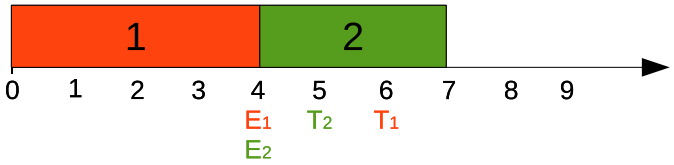


Figura 2.2: Possível programação π de $I = \{1, 2\}$, na qual $\max_{x\in I} C_x^\pi = 7$ e $f(\pi) = 2 \times \beta_2 = 8$.

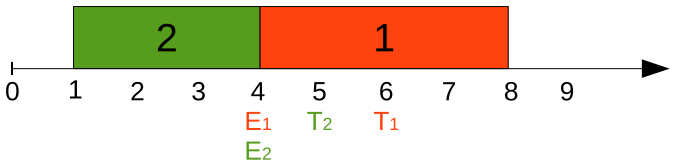


Figura 2.3: Programação ótima π^* de $I = \{1, 2\}$. Note que $\max_{x\in I} C_x^{\pi^*} = 8$ e $f(\pi^*) = 2 \times \beta_1 = 2$.

Seja $S_{\max}^I$ o maior tempo total de preparação da máquina possível contido em uma programação de I. O problema de determinação de S_{max}^I pode ser facilmente reduzido a uma variação do problema do caixeiro viajante, denominada *Maximum Traveling Salesman Problem with General Non-Negative weights* – MAX TSP (Pinedo, 2012; Gutin & Punnen, 2007). Com esse propósito, seja x_0 uma tarefa fictícia tal que $S_{x,x_0} = S_{x_0,x} = 0, \forall x \in I$. Cada tarefa em $I \cup \{x_0\}$ corresponde a uma cidade no MAX TSP e a distância da cidade x para a cidade y é dada pelo tempo de preparação S_{xy}. Observa-se que, neste caso, a distância entre duas cidades não é necessariamente simétrica. Seja γ_x a variável de decisão que diz a ordem em que a cidade x é visitada, $\forall x \in I \cup \{x_0\}$. Assim como em Miller *et al.* (1960), sejam u_{xy} variáveis de decisão binárias tais que, $\forall x, y \in I \cup \{x_0\}$ e $x \neq y$:

$$u_{xy} = \begin{cases} 1, & \text{se a cidade } y \text{ é visitada imediatamente após a cidade } x; \\ 0, & \text{caso contrário.} \end{cases}$$

O problema de determinar $S^I_{\max}$ pode ser formulado pela função objetivo (2.4) e pelas restrições (2.5)-(2.10), como segue:

$$\max \quad \sum_{x\in I}\sum_{y\in I\setminus\{x\}} S_{xy}\cdot u_{xy} \tag{2.4}$$

$$\text{s. a.} \quad \sum_{\substack{y\in I\cup\{x_0\}\\ y\neq x}} u_{xy} = 1 \quad \forall\, x\in I\cup\{x_0\} \tag{2.5}$$

$$\sum_{\substack{x\in I\cup\{x_0\}\\ x\neq y}} u_{xy} = 1 \quad \forall\, y\in I\cup\{x_0\} \tag{2.6}$$

$$\gamma_0 = 1 \tag{2.7}$$

$$\gamma_x + (n+1)u_{xy} - n \leq \gamma_y \quad \forall\, x,\, y\in I\cup\{x_0\} \quad \text{e} \quad x\neq y \tag{2.8}$$

$$\gamma_x \geq 0 \quad \forall\, x\in I \tag{2.9}$$

$$u_{xy} \in \{0,\,1\} \quad \forall\, x,y\in I\cup\{x_0\} \tag{2.10}$$

A função objetivo, dada pela expressão (2.4), busca a maximização do tempo total de preparação da máquina (ou, equivalentemente, a maior distância percorrida). As restrições (2.5) e (2.6) garantem que uma preparação é realizada antes e outra depois de cada tarefa em $I\cup\{x_0\}$ (ou, analogamente, o caixeiro deve sempre sair de uma cidade para chegar em outra). A restrição (2.7) impõe que a tarefa fictícia x_0 seja a primeira a ser executada (isto é, a cidade x_0 é o ponto de partida do caixeiro viajante). As restrições (2.8) previnem sub-rotas. As restrições (2.9) e (2.10) dizem respeito aos domínios das variáveis de decisão.

A Proposição 2.2 fornece um limite superior para $S^I_{\max}$. Esse limite é uma alternativa para os casos em que a resolução do modelo anterior é inviável.

Proposição 2.2

$$S^I_{\max} \leq \min\left(\underbrace{\sum_{x\in I}\max_{y\in I\setminus\{x\}} S_{yx} - \min_{x\in I}\max_{y\in I\setminus\{x\}} S_{yx}}_{UB_1},\ \underbrace{\sum_{x\in I}\max_{y\in I\setminus\{x\}} S_{xy} - \min_{x\in I}\max_{y\in I\setminus\{x\}} S_{xy}}_{UB_2}\right).$$

Prova: O maior tempo total de preparação da máquina antes da execução da tarefa $x\in I$ é $\max_{y\in I\setminus\{x\}} S_{yx}$. Dado que não é necessário preparar a máquina antes da execução da primeira tarefa, o número de preparações da máquina em uma programação de n tarefas é $n-1$. Logo, UB_1 é um limite superior para S^I_{max}.

Analogamente, o maior tempo de preparação da máquina após a execução da tarefa $x\in I$ é $\max_{y\in I\setminus\{x\}} S_{xy}$. Uma vez que o número de preparações da máquina é $n-1$, UB_2 também é um limite superior sobre S^I_{max}. Portanto, $S^I_{max}\leq \min(UB_1, UB_2)$. □

A seguinte definição particiona o conjunto das programações de tarefas em I.

Definição 2.1 (Programação natural) *Uma programação possível π de I é dita natural se, mantendo a ordem de execução das tarefas, não for possível construir uma programação π' tal que $f(\pi')\leq f(\pi)$ e uma das seguintes condições seja satisfeita:*

- $\max_{x\in I} C^{\pi'}_x < \max_{x\in I} C^{\pi}_x$;
- $\max_{x\in I} C^{\pi'}_x - \min_{x\in I} s^{\pi'}_x < \max_{x\in I} C^{\pi}_x - \min_{x\in I} s^{\pi}_x$.

Em outras palavras, uma programação π é dita natural se, fixada a sequência de execução das tarefas, toda programação com custo associado menor ou igual ao custo de π for concluída depois de π e possui tempo de execução maior que o de π. Para os problemas de programação em que não há penalidades por antecipação, a definição de programação natural proposta é equivalente à definição de programação semi-ativa de Pinedo (2012).

A Proposição 2.3 permite reduzir o espaço de busca do SMSPETP ao conjunto de programações naturais.

Proposição 2.3 *Existe uma programação ótima de I que é natural.*

Prova: Seja π uma programação ótima de I que não é natural. Logo, existe uma outra programação ótima π' tal que $\max_{x\in I} C_x^{\pi'} - \min_{x\in I} s_x^{\pi'} < \max_{x\in I} C_x^{\pi} - \min_{x\in I} s_x^{\pi}$ ou $\max_{x\in I} C_x^{\pi'} < \max_{x\in I} C_x^{\pi}$. Se π' não é natural, então existe uma terceira programação ótima cujo tempo de execução é menor que o respectivo tempo de π' ou cuja data de conclusão é mais cedo que a data de π'. Já que a ociosidade de máquina é reduzida cada vez que esse procedimento é realizado, ele pode ser repetido até que uma programação ótima natural de I seja encontrada. □

Lema 2.1 *Dada uma programação natural π de I, se a última tarefa em π (x_n) tem a sua execução iniciada após a data* $\max\big(\max_{x\in I}(E_x - P_x), 0\big)$*, então não existe ociosidade de máquina entre essa data e a data de início de x_n* $\big(s_{x_n}^{\pi}\big)$.

Prova: Suponha que a execução de x_n se inicia após a data $\max\big(\max_{x\in I}(E_x - P_x), 0\big)$ e que exista ociosidade de máquina entre essa data e $s_{x_n}^{\pi}$. Toda tarefa y que tem sua execução iniciada após a data $\max\big(\max_{x\in I}(E_x - P_x), 0\big)$ é tal que

$$C_y^{\pi} = s_y^{\pi} + P_y \geq \max_{x\in I}\big(E_x - P_x\big) + P_y \geq \big(E_y - P_y\big) + P_y = E_y$$

Logo, essas tarefas não estão antecipadas e podem ser deslocadas para a esquerda (isto é, serem adiantadas), mantendo-as não antecipadas, até que não exista mais ociosidade de máquina entre $\max\big(\max_{x\in I}(E_x - P_x), 0\big)$ e $s_{x_n}^{\pi}$. A programação obtida π' é tal que $f(\pi') \leq f(\pi)$ e $\max_{x\in I} C_x^{\pi'} < \max_{x\in I} C_x^{\pi}$, o que contradiz o fato de π ser uma programação natural. □

A Proposição 2.4 fornece um limite superior para o horizonte de planejamento das tarefas de I.

Proposição 2.4 *Se π é uma programação natural de I, então*

$$\max_{x\in I} C^{\pi} \leq \max\Big(\max_{x\in I}(E_x - P_x),\, 0\Big) + \sum_{x\in I} P_x + S^I_{\max}.$$

Prova: Suponha que π é uma programação natural de I. De acordo com o Lema 2.1, um limite superior sobre a data de conclusão de π $\big(\max_{x\in I} C_x^{\pi}\big)$ é obtido ao se programar todas as tarefas em I consecutivamente, sem ociosidade de máquinas e iniciando a primeira tarefa na data $\max\big(\max_{x\in I}(E_x - P_x), 0\big)$. □

O limite superior fornecido pela Proposição 2.4 garante que existe tempo suficiente para executar todas as tarefas sem antecipações, independentemente da sequência de execução dessas tarefas.

Lema 2.2 *Dada uma programação natural π de I, se a primeira tarefa em π (x_1) for concluída antes da data* $\min_{x\in I} T_x$*, então não existe ociosidade de máquina entre a conclusão de x_1* $\big(C_{x_1}^{\pi}\big)$ *e* $\min_{x\in I} T_x$.

Prova: Suponha que a tarefa x_1 é concluída antes da data $\min_{x\in I} T_x$ e que exista ociosidade de máquina entre $C^{\pi}_{x_1}$ e $\min_{x\in I} T_x$. Uma vez que $C^{\pi}_y \leq \min_{x\in I} T_x \leq T_y$ para toda tarefa y concluída antes de $\min_{x\in I} T_x$, essas tarefas não estão atrasadas e podem ser deslocadas para a direita (isto é, serem adiadas), mantendo-as não atrasadas, até que não exista mais ociosidade de máquina entre $C^{\pi}_{x_1}$ e $\min_{x\in I} T_x$. A programação π' obtida após esse deslocamento é tal que $f(\pi') \leq f(\pi)$ e $\max_{x\in I} C^{\pi'}_x - \min_{x\in I} s^{\pi'}_x < \max_{x\in I} C^{\pi}_x - \min_{x\in I} s^{\pi}_x$, o que contradiz o fato de π ser uma programação natural. □

A Proposição 2.5 fornece um limite inferior para o horizonte de planejamento das tarefas de I.

Proposição 2.5 *Se π é uma programação natural de I, então a data de início de π $\left(\min_{x\in I} s^{\pi}_x\right)$ é tal que:*

$$\min_{x\in I} s^{\pi}_x \geq \max\left(\min_{x\in I} T_x - \sum_{x\in I} P_x - S^I_{max},\, 0\right).$$

Prova: Seja π uma programação natural de I. Com base no Lema 2.2, um limite inferior para a data de início de π $\left(\min_{x\in I} s^{\pi}_x\right)$ pode ser obtido programando-se todas as tarefas em I consecutivamente, sem ociosidade de máquina e iniciando-se a primeira tarefa na data $\min_{x\in I} T_x - \sum_{x\in I} P_x - S^I_{max}$ ou na data zero. □

O limite inferior dado pela Proposição 2.5 assegura que existe tempo suficiente para executar todas as tarefas em I sem atrasos, ou a primeira tarefa é iniciada na data zero, independentemente da ordem de execução dessas tarefas.

O teorema seguinte fornece um horizonte de planejamento que contém uma programação ótima das tarefas de I. Ele é uma consequência direta das Proposições 2.3, 2.4 e 2.5.

Teorema 2.1 *Existe uma programação ótima π^* de I tal que*

$$\max\left(\min_{x\in I} T_x - \sum_{x\in I} P_x - S^I_{max},\, 0\right) \leq \min_{x\in I} s^{\pi^*}_x \leq \max_{x\in I} C^{\pi^*}_x \leq \max\left(0, \max_{x\in I}(E_x - P_x)\right) + \sum_{x\in I} P_x + S^I_{max}.$$

Lema 2.3 *A data mais cedo para o início da execução de uma dada tarefa x em programações naturais de I ocorre em pelo menos uma programação na qual x é a primeira tarefa a ser executada.*

Prova: Seja π uma programação natural de I em que uma dada tarefa x não é a primeira a ser executada. Considere a programação π' oriunda de π pela realocação das tarefas executadas antes de x para após a última tarefa programada em π. Caso π' não seja natural, é possível obter uma programação natural π'' de π'. A programação π'' é tal que x é a primeira tarefa executada e $s^{\pi''}_x \leq s^{\pi'}_x = s^{\pi}_x$. □

Corolário 2.1 *A data mais tarde para o início da execução de uma dada tarefa x em programações naturais de I ocorre em pelo menos uma programação na qual x é a última tarefa executada.*

Prova: Seja π uma programação natural de I em que uma dada tarefa x não é a última a ser executada. A programação π' obtida de π pela realocação das tarefas executadas após x para imediatamente antes da primeira tarefa programada em π é uma programação natural tal que x é a última tarefa a ser executada e $s^{\pi'}_x \geq s^{\pi}_x$. Observa-se que pode ser necessário adiar a execução das demais tarefas para que essa realocação seja possível. □

O próximo teorema fornece um horizonte de planejamento para cada tarefa $x \in I$.

Teorema 2.2 *Se π é uma programação natural de I, então, para qualquer tarefa $x \in I$, tem-se:*

$$s_x^{LB} \leq s_x^{\pi} \leq s_x^{UB},$$

em que:

- $s_x^{LB} = \max\left(0, \min\left(\min_{y \in I\setminus\{x\}} T_y - \sum_{y \in I} P_y - S_{\max}^{I\setminus\{x\}} - \max_{y \in I\setminus\{x\}} S_{xy}, T_x - P_x\right)\right)$ *e*
- $s_x^{UB} = \max\left(\max\left(0, \max_{y \in I\setminus\{x\}} (E_y - P_y)\right) + \sum_{y \in I\setminus\{x\}} P_y + S_{max}^{I\setminus\{x\}} + \max_{y \in I\setminus\{x\}} S_{yx}, E_x - P_x\right)$.

Prova: Dada uma tarefa $x \in I$, de acordo com o Lema 2.3, a data mais cedo de início de x em programações naturais de I ocorre em uma programação π' na qual x é a primeira tarefa a ser executada. Neste caso, $\min_{y \in I} s_y^{\pi'} = s_x^{\pi'}$ e o maior tempo de preparação da máquina contido em π' é $S_{\max}^{I\setminus\{x\}} + \max_{y \in I\setminus\{x\}} S_{xy}$. Se $s_x^{\pi'} < s_x^{LB}$, então é possível adiar a execução da tarefa x, mantendo a sequência de execução das tarefas, sem aumentar o custo associado a π' (da mesma forma em que foi realizado na prova do Lema 2.2, mas, neste caso, sabe-se que a tarefa x é a primeira a ser executa em π'). Isto contradiz o fato de π' ser uma programação natural. Logo, se π é uma programação natural de I, então $s_x^{LB} \leq s_x^{\pi'} \leq s_x^{\pi}$.

Similarmente, com base no Corolário 2.1, a data mais tarde de início da tarefa x em programações naturais de I ocorre em uma programação π'' na qual x é a última tarefa a ser executada. Neste caso, $\max_{x \in I} C_x^{\pi''} = C_x^{\pi''} = s_x^{\pi''} + P_x$ e o maior tempo de preparação da máquina contido em π'' é $S_{\max}^{I\setminus\{x\}} + \max_{y \in I\setminus\{x\}} S_{yx}$. Se $s_x^{\pi''} > s_x^{UB}$, então é possível adiantar a execução da tarefa x, mantendo a sequência de execução das tarefas, sem aumentar o custo associado a π'' (da mesma forma em que foi feito na prova do Lema 2.1, mas, neste caso, sabe-se que x é a última tarefa a ser executada em π''). Isto contradiz o fato de π'' ser uma programação natural. Logo, se π é uma programação natural de I, então $s_x^{\pi} \leq s_x^{\pi''} \leq s_x^{UB}$. □

A Subseção 2.3.1 compara o limite superior para o tempo máximo de preparação da máquina contido em uma programação de I, dado pela Proposição 2.2, com o valor do respectivo tempo máximo, ao passo que a Subseção 2.4 conclui esta Seção.

2.3.1 Resultados computacionais

Nesta seção é realizada uma breve análise do limite superior fornecido pela Proposição 2.2, aqui denotado por S_{UB}^I, para o tempo máximo de preparação da máquina contido em uma programação de I ($S_{\max}^I$).

Os experimentos computacionais foram realizados com os conjuntos de problemas-teste com 10, 20, 30, 40, 50 e 75 tarefas descritos na Seção 2.2. Os valores de $S_{\max}^I$ foram obtidos por meio da formulação de programação matemática dada pelas Equações (2.4) – (2.10). Tal formulação foi implementada e resolvida com a *C++ Concert Technology* e com o otimizador *IBM ILOG CPLEX Optimization Studio 12.6.2*. Por outro lado, os valores de S_{UB}^I foram calculados por meio de um algoritmo implementado na linguagem C++, sendo utilizado o compilador g++, versão 4.8.5, para a sua execução.

Utilizou-se um computador Intel® Xeon(R) CPU E5620 @ 2.40GHz × 16, com 48 GB de RAM e sistema operacional CentOS Linux 7 na realização dos experimentos. O CPLEX foi configurado para utilizar somente uma *thread* e os demais parâmetros não foram alterados. Apesar de o processador do equipamento utilizado possuir mais de um núcleo, os algoritmos não foram otimizados para multiprocessamento.

Seja I o conjunto de tarefas em um dado problema-teste. A qualidade de S^I_{UB} é medida pelo seu desvio em relação a $S^I_{\max}$, dado por:

$$DR_I = \frac{S^I_{UB} - S^I_{\max}}{S^I_{\max}} \times 100\%. \tag{2.11}$$

Deste modo, quanto menor for o valor de DR_I, mais próximo de $S^I_{\max}$ estará o limite superior dado pela Proposição 2.2 para as tarefas de I.

A Tabela 2.1 apresenta um resumo dos resultados obtidos. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. Para cada conjunto de problemas, as colunas "$\overline{\text{tempo}}$" apresentam as médias dos tempos demandados (em segundos) para encontrar os valores de S^I_{UB} e de $S^I_{\max}$, respectivamente. A coluna "$\overline{DR_I}$" apresenta os desvios médios de S^I_{UB} em relação aos respectivos valores de $S^I_{\max}$ (em porcentagem).

Tabela 2.1: Resultados obtidos ao se utilizar a Proposição 2.2 para determinar S^I_{UB}.

n	$\overline{\text{tempo}}$ (s)		$\overline{DR_I}$ (%)
	S^I_{UB}	$S^I_{\max}$	
10	0,00	0,01	1,80
20	0,00	0,04	0,85
30	0,00	0,08	0,28
40	0,00	0,22	0,04
50	0,00	0,47	0,01
75	0,00	1,92	0,00

De acordo com a Tabela 2.1, os tempos médios necessários para encontrar os valores de S^I_{UB} foram sempre próximos de zero segundos, ao passo que os tempos médios demandados para encontrar os valores de $S^I_{\max}$ foram maiores ou iguais a 0,01 segundos. Para os problemas-teste com 75 tarefas, por exemplo, a média dos tempos demandados para encontrar os valores de $S^I_{\max}$ foi igual a 1,92 segundos. Ainda na Tabela 2.1, pode-se ver que os desvios médios de S^I_{UB} em relação aos respectivos valores de $S^I_{\max}$ foram sempre menores ou iguais a 1,80%. Os valores desses desvios decrescem à medida em que se aumenta o número de tarefas, chegando a ser iguais a 0,00% para os problemas-teste com 75 tarefas.

2.4 Conclusões do capítulo

Este capítulo descreveu detalhadamente o problema de programação de tarefas tratado nesta tese, o SMSPETP (*Single Machine Scheduling Problem with Earliness and Tardiness Penalties*), bem como as classes desse problema que foram consideradas, isto é, o SMSPETP-SDS (SMSPETP com tempos de preparação da máquina dependentes da sequência de execução das tarefas) e o SMSPETP-SIS (SMSPETP com tempos de preparação da máquina independentes da sequência de execução das tarefas). Ademais, foram propostos resultados teóricos que permitem estabelecer um horizonte de planejamento para a execução de cada tarefa do SMSPETP. O horizonte de planejamento proposto assegura que a solução ótima do problema esteja contido nele. Observa-se que, de acordo com a pesquisa realizada, este é o primeiro trabalho que faz um estudo do horizonte de planejamento para as tarefas do SMSPETP-SDS.

Um dos parâmetros utilizados na determinação do horizonte de planejamento proposto é o tempo máximo de preparação da máquina contido em qualquer programação de I, denotado por $S^I_{\max}$. O valor de $S^I_{\max}$ é determinando resolvendo-se um problema de programação matemática linear inteiro misto. Além disso, foi proposto um limite superior para $S^I_{\max}$. Esse limite superior,

denotado por S^I_{UB}, é uma alternativa para os casos em que a resolução do modelo matemático proposto é inviável. Resultados computacionais realizados com problemas-teste com até 75 tarefas mostram que esse limite é próximo de $S^I_{\max}$ e decresce à medida em que se aumenta o número de tarefas a serem executadas. Além disso, o tempo demandado pelo CPLEX para resolver a formulação proposta para determinar $S^I_{\max}$ é muito superior ao tempo necessário para calcular o respectivo limite superior proposto.

O horizonte de planejamento para a execução das tarefas do SMSPETP-SDS é muito importante na determinação dos parâmetros de entrada das formulações matemáticas que representam esse problema. Logo, o horizonte de planejamento proposto nesta seção é utilizado para definir os parâmetros de entrada de todas formulações matemáticas para o SMSPETP-SDS apresentadas no Capítulo 4.

Capítulo 3

Algoritmos propostos

Neste capítulo são propostos algoritmos heurísticos para a resolução do SMSPETP-SDS, bem como um algoritmo exato para o SMSPETP-SIS. Na Seção 3.1 são apresentados três algoritmos de programação ótima de uma dada sequência de tarefas. O primeiro deles é um algoritmo da literatura, o segundo é uma contribuição deste trabalho e o terceiro é uma proposta de melhoria redução da complexidade computacional do algoritmo da literatura. A Seção 3.2 descreve a metaheurística Busca em Vizinhança Variável (*Variable Neighborhood Search* – VNS) e descreve algoritmos propostos neste trabalho baseados nessa metaheurística para resolver o SMSPETP-SDS. A Seção 3.3 apresenta uma técnica de Enumeração Implícita (EI) e descreve um algoritmo baseado nessa técnica proposto para resolver o SMSPETP-SIS. Na Seção 3.4 é feita a apresentação e análise dos resultados obtidos com os algoritmos propostos. Por fim, a Seção 3.5 conclui este capítulo.

3.1 Programação de uma dada sequência de tarefas

O objetivo do SMSPETP pode ser dividido em dois subobjetivos que devem ser buscados simultaneamente. Um deles é determinar a sequência de execução das tarefas, isto é, a ordem em que as tarefas devem ser executadas. O outro é determinar a programação ótima dessa sequência, ou seja, determinar a data de início de cada tarefa na sequência de execução, de modo que a soma ponderada das antecipações e atrasos das tarefas seja a menor possível.

A Figura 3.1 ilustra duas programações possíveis para uma dada sequência de execução de três tarefas. Na programação π_1, as datas de início de execução das tarefas foram programadas de modo que a tarefa 3 é concluída antecipadamente, acarretando em penalidade por antecipação, e a tarefa 2 é concluída dentro de sua respectiva janela de conclusão, não gerando penalidades. Já na programação π_2, as datas de início das tarefas foram programadas de forma que a tarefa 2 é concluída com atraso, gerando penalidade, e a tarefa 3 é concluída dentro de sua janela de conclusão, não agregando penalidade. Nessa figura também são ilustrados os tempos de preparação e de ociosidade da máquina. De acordo com os valores das penalidades incorridas, uma dessas programações será melhor que a outra nessa sequência. Por exemplo, considerando que o tempo de antecipação da tarefa 3 na programação π_1 é igual ao tempo de atraso da tarefa 2 na programação π_2, a programação π_2 será mais vantajosa que a programação π_1, caso $\alpha_3 > \beta_2$, e, caso contrário, a programação π_1 será mais vantajosa que a programação π_2.

Os trabalhos encontrados na literatura tratam o SMSPETP com procedimentos heurísticos que, iterativamente, geram sequências de execução das tarefas por meio de regras de prioridade ou procedimentos de busca. Para avaliar cada sequência gerada, é necessário determinar a programação ótima dessa sequência, permitindo a ociosidade de máquina entre a execução de tarefas consecutivas. Por se tratar de um procedimento realizado um elevado número de vezes,

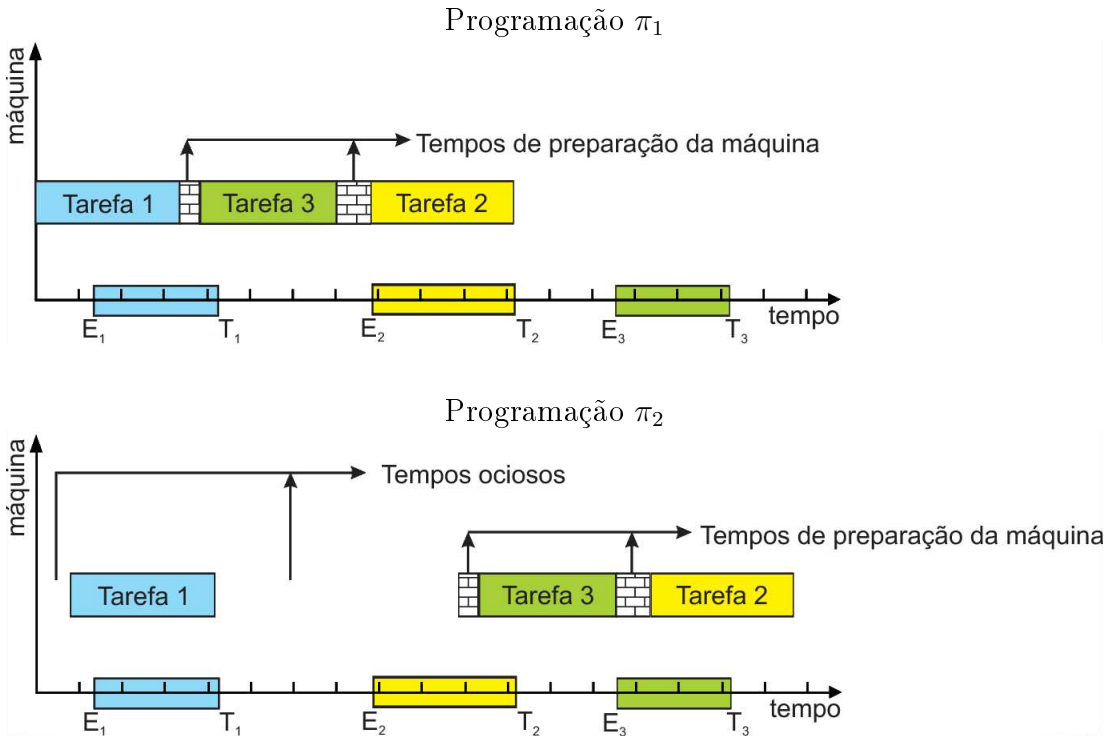


Figura 3.1: Duas possíveis programações com a mesma sequência de execução de três tarefas.

é fundamental o uso de um algoritmo capaz de determinar a programação ótima de uma dada sequência de tarefas o mais rapidamente possível.

Fry *et al.* (1987) propõem uma formulação de programação matemática para determinar uma programação ótima de uma dada sequência de tarefas com datas de conclusão distintas. Garey *et al.* (1988) apresentam um algoritmo de complexidade $O(n \log n)$ que encontra uma programação ótima com datas de conclusão distintas e penalidades por antecipações e atrasos não ponderados.

Yano & Kim (1991) sugerem um algoritmo de programação dinâmica para determinar uma programação ótima de uma dada sequência de tarefas com datas de conclusão distintas e com penalidades por unidade de antecipação menores ou iguais às respectivas penalidades por unidade de atraso. Szwarc & Mukhopadhyay (1995) propõem um algoritmo baseado no conceito de *clusters*, que determina uma programação ótima de uma dada sequência de tarefas com datas de conclusão distintas e pesos genéricos para as penalidades por unidade de antecipação e atraso, enquanto Davis & Kanet (1993) e Lee & Choi (1995) apresentam algoritmos para a programação ótima de sequências dessa mesma classe de problemas. São encontrados mais trabalhos sobre problemas de programação de tarefas com possibilidade de alocação de tempos ociosos em Kanet & Sridharan (2000) e Józefowska (2007).

Quando se trata do problema de programação de tarefas com janelas de conclusão distintas, os procedimentos de programação ótima de uma dada sequência de tarefas são limitados aos trabalhos de Koulamas (1996), Wan & Yen (2002) e Gomes Júnior *et al.* (2007). Em Koulamas (1996), é apresentado um algoritmo de complexidade polinomial para problemas com penalidades por antecipação e atraso não ponderadas. Esse algoritmo é uma extensão do trabalho de Garey *et al.* (1988). Wan & Yen (2002) estendem os trabalhos de Davis & Kanet (1993), Lee & Choi (1995) e Szwarc & Mukhopadhyay (1995) para o SMSPETP-SIS. Por último, Gomes Júnior *et al.* (2007) adaptam o algoritmo de Wan & Yen (2002) para considerar tempos de preparação da máquina dependentes da sequência de execução das tarefas.

O algoritmo de programação ótima de uma dada sequência de tarefas do SMSPETP-SIS de Wan & Yen (2002), o qual foi adaptado por Gomes Júnior *et al.* (2007) para o SMSPETP-SDS,

é reproduzido na Seção 3.1.1. Ademais, um novo algoritmo de alocação ótima de tempos ociosos em uma dada sequência de execução do SMSPETP-SDS é proposto na Seção 3.1.2, enquanto uma melhoria do algoritmo de datas ótimas de Wan & Yen (2002) é proposta na Seção 3.1.3.

3.1.1 *Optimal Timing Algorithm* – OTA

Nesta seção é reproduzido o algoritmo de programação ótima de uma dada sequência de tarefas do SMSPETP-SIS proposto por Wan & Yen (2002), denotado por OTA (*Optimal Timing Algorithm*). É apresentada a versão do OTA adaptada por Gomes Júnior *et al.* (2007) para o SMSPETP-SDS, isto é, a versão do OTA que considera a existência de tempos de preparação da máquina dependentes da sequência de execução das tarefas.

Dada uma sequência $X = (x_1, x_2, \ldots, x_n)$ das tarefas de I, seja π_k uma programação parcial das k primeiras tarefas em X e $C_{x_i}^{\pi_k}$ a data de conclusão da tarefa x_i em π_k. Suponha que um subconjunto de tarefas $B = \{x_u, x_{u+1}, \ldots, x_v\}$ forma um bloco na programação π_k, isto é, as tarefas de B são programadas sem ociosidade de máquina entre elas. Além disso, assuma que existem l blocos em π_k e que $prim(B)$ e $ult(B)$ representam a primeira e a ultima tarefa do bloco B, respectivamente. A programação parcial inicial π_1 é obtida programando a tarefa x_1 para ser concluída na data E_{x_1}, se $P_{x_1} \leq E_{x_1}$, ou na data P_{x_1}, se $P_{x_1} > E_{x_1}$. Dada a programação π_k, a tarefa x_{k+1} é programada como segue. Seja $S_{(x_k)(x_{k+1})}$ o tempo de preparação da máquina entre a execução das tarefas consecutivas x_k e x_{k+1}. Se $C_{x_k}^{\pi_k} + S_{(x_k)(x_{k+1})} + P_{x_{k+1}} < E_{x_{k+1}}$, então a tarefa x_{k+1} é programada para ser concluída na data $E_{x_{k+1}}$, isto é, a tarefa x_{k+1} é programada para ser concluída no início da sua janela de conclusão e dá origem a um novo bloco. Por outro lado, se $C_{x_k}^{\pi_k} + S_{(x_k)(x_{k+1})} + P_{x_{k+1}} \geq E_{x_{k+1}}$, então a tarefa x_{k+1} é programada para ser iniciada na data $C_{x_k}^{\pi_k} + S_{(x_k)(x_{k+1})}$, ou seja, x_{k+1} é incluída no último bloco de π_k.

Após determinar a data de conclusão de cada tarefa, deve-se rever a programação dos blocos. O custo associado à tarefa x na programação π_k é dado por:

$$g_x(C_x^{\pi_k}) = \alpha_x \cdot \max(0, E_x - C_x^{\pi_k}) + \beta_x \cdot \max(0, C_x^{\pi_k} - T_x).$$

Logo, o custo associado às tarefas de B após deslocar esse bloco por φ unidades de tempo para a esquerda é dado por:

$$Custo_B(\varphi) = \sum_{x \in B} g_x(C_x^{\pi_k} - \varphi), \tag{3.1}$$

em que $C_x^{\pi_k} - \varphi$ corresponde à nova data de conclusão da tarefa x. O resultado a seguir analisa a topologia dessa função.

Lema 3.1 ***(Wan & Yen, 2002):*** *A soma de duas funções lineares por partes e convexas é também uma função linear por partes e convexa.*

Com base no Lema 3.1, tem-se:

Proposição 3.1 ***(Wan & Yen, 2002):*** *$Custo_B(\varphi)$ é uma função linear por partes e convexa em relação a φ.*

Devido à natureza linear por partes e convexa da função custo, o custo mínimo de um bloco B ocorre em um dos pontos extremos da função $Custo_B$, isto é, ou no início ou no final da janela de conclusão de uma das tarefas no bloco. Logo, o custo mínimo pode ser facilmente obtido pela comparação dos somatórios das penalidades das tarefas em B, sendo cada um desses somatórios obtido ao programar uma tarefa em B para ser concluída no início ou no final de sua respectiva janela de conclusão. Todos os possíveis somatórios são calculados e o menor deles está associado ao ponto de custo mínimo de B. Em caso de empate, escolhe-se o menor ponto de mínimo. Assim que o ponto de mínimo é encontrado, o bloco B é deslocado na direção desse ponto até que um dos três casos seguintes aconteça:

(i) A primeira tarefa do bloco B é iniciada na data 0;

(ii) o bloco B é programado em seu ponto de mínimo;

(iii) O bloco B se junta ao bloco antecessor, ou seja, $C^{\pi_k}_{prim(B)}$ torna-se igual a $C^{\pi_k}_{prim(B)-1} + S_{prim(B)-1,prim(B)} + P_{prim(B)}$.

Esse procedimento de verificação da programação de um bloco é realizado até que ocorra o caso (i) ou (ii) para cada bloco de π_k.

Proposição 3.2 ***(Wan & Yen, 2002):*** *O custo total associado a uma dada sequência de tarefas é ótimo se cada bloco B na programação de X estiver no seu ponto de mínimo, exceto o primeiro bloco, que pode ter sua execução programada para iniciar na data zero.*

Para exemplificar o procedimento OTA, considere o SMSPETP-SDS de 4 tarefas representado pela Tabela 3.1. Nessa tabela, a primeira coluna apresenta o conjunto de tarefas a serem programadas, isto é, $I = \{1, 2, 3, 4\}$. As demais colunas indicam o tempo de execução (P_x), o custo por unidade de antecipação (α_x), o custo por unidade de atraso (β_x), a data de início (E_x) e de término (T_x) da janela de conclusão, bem como os tempos de preparação da máquina relativos a cada tarefa $x \in I$.

Tabela 3.1: Problema-teste para exemplificar o procedimento OTA.

I	P_x	E_x	T_x	α_x	β_x	S_{xy}			
						1	2	3	4
1	3	14	15	2	4	0	2	1	2
2	4	22	24	7	9	1	0	2	3
3	4	9	12	7	8	1	3	0	1
4	3	5	7	1	4	1	2	2	0

Dada a sequência $X = (3, 4, 1, 2)$, a primeira tarefa, $x_1 = 3$, é programada para ser concluída na data $C^{\pi_1}_{x_1} = C^{\pi_1}_3 = \max(E_3, P_3) = E_3 = 9$, conforme a Figura 3.2.

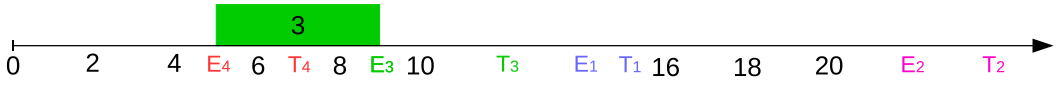


Figura 3.2: Programação da primeira tarefa de X.

A segunda tarefa, $x_2 = 4$, é programada para ser concluída na data $C^{\pi_2}_{x_2} = C^{\pi_2}_4 = 13$ (Figura 3.3), pois $C^{\pi_1}_3 + S_{34} + P_4 \geq E_4$ $(13 > 5)$. Neste momento, é necessário verificar se o bloco formado, $\{x_1, x_2\} = \{3, 4\}$, está no seu ponto de mínimo. Para isto, primeiro determina-se as possíveis datas de início da execução do bloco. Essas datas são obtidas programando cada tarefa do bloco para ser concluída no início e no final de sua janela de conclusão (visto que o custo mínimo do bloco se dá no início ou no final da janela de conclusão de uma das tarefas no bloco). No exemplo em questão, as possíveis datas de início da execução do bloco $\{3, 4\}$ são 8, 5 e 0. Os custos totais associados ao bloco B caso a sua execução seja iniciada nessas datas são, respectivamente, 36, 24 e 39. Logo, o bloco B terá custo mínimo se sua execução for iniciada na data 5. Como o bloco B já encontra-se programado no seu ponto de mínimo, passa-se para a alocação da terceira tarefa da sequência X.

A terceira tarefa, $x_3 = 1$, é alocada para ser concluída na data $C^{\pi_3}_{x_3} = C^{\pi_3}_1 = 17$ (Figura 3.4), pois $C^{\pi_2}_4 + S_{41} + P_1 \geq E_1$ $(17 > 14)$. Neste momento, é necessário verificar se o bloco formado, $\{x_1, x_2, x_3\} = \{3, 4, 1\}$, está no seu ponto de mínimo. As datas candidatas para o

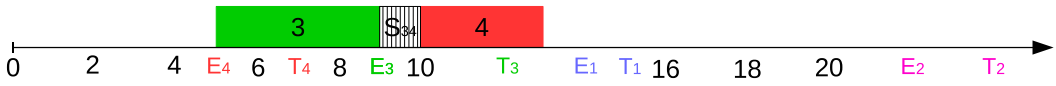


Figura 3.3: Alocação da segunda tarefa na programação da sequência X.

início da execução desse bloco são 8, 5, 3, 2 e 0. Os custos totais associados ao bloco B caso a sua execução seja iniciada nessas datas são, respectivamente, 56, 32, 30, 33 e 43. Como o bloco B encontra-se programado para iniciar sua execução na data 5 e o ponto de mínimo desse bloco é a data 3 (com custo associado igual a 30), o bloco B reprogramado para ter sua execução no ponto de mínimo (ou seja, data 3), conforme ilustrado na Figura 3.5.

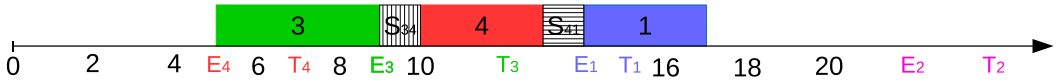


Figura 3.4: Alocação da terceira tarefa na programação da sequência X.

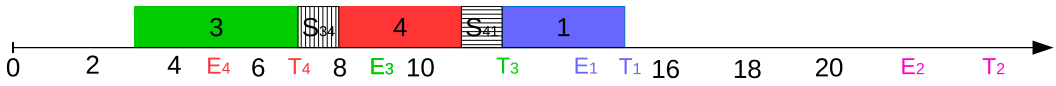


Figura 3.5: Bloco $\{x_1, x_2, x_3\} = \{3, 4, 1\}$ em seu ponto de mínimo.

Finalmente, a última tarefa, $x_4 = 2$, é programada para ser concluída na data $C_{x_4}^{\pi_4} = C_2^{\pi_4} = 22$ (Figura 3.6), pois $C_1^{\pi_3} + S_{12} + P_2 < E_2$ $(21 < 22)$. Observa-se que um novo bloco, composto apenas pela tarefa $x_4 = 2$, é criado. Esse bloco encontra-se em seu ponto de mínimo, pois a tarefa $x_4 = 2$ não acarreta em nenhuma penalidade. Portanto, tem-se que a programação representada pela Figura 3.6 é ótima para a sequência $X = (3, 4, 1, 2)$ e o menor custo associado a essa sequência é 30.

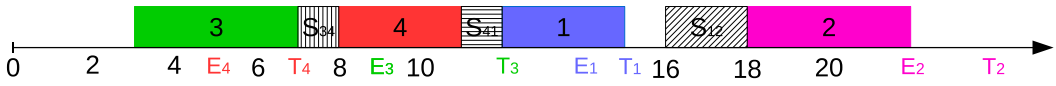


Figura 3.6: Programação ótima da sequência X.

3.1.1.1 Análise de complexidade do algoritmo OTA

O Algoritmo 3.1 reproduz o procedimento OTA aplicado a uma dada sequência de tarefas $X = \{x_1, x_2, \ldots, x_n\}$. Nele, o procedimento *DeslocaBloco*(B) desloca o bloco B até o seu ponto de mínimo, até que $C_{x_1}^{\pi_i} = P_{x_1}$ ou até B se unir ao bloco antecessor.

O custo computacional associado às linhas 1-3 do Algoritmo 3.1 é constante. Para cada iteração $i \in \{2, 3, \ldots, n\}$ do laço "para" da linha 4, a complexidade associada às linhas 5-10 também é constante. De acordo com o que foi apresentado por Wan & Yen (2002) e implementado por Gomes Júnior *et al.* (2007), a determinação do ponto de mínimo do bloco B_l envolve o cálculo de $2 \cdot |B_l|$ somatórios de tamanho $|B_l|$, em que $|B_l|$ representa a cardinalidade de B_l. Logo, o custo computacional de se determinar o ponto de mínimo de B_l é $O(|B_l|^2)$. A maior cardinalidade possível de B_l na iteração i é i (ou seja, $|B_l| \leq i$, $\forall i \in \{2, 3, \ldots, n\}$). Consequentemente, a complexidade computacional de se verificar a condição do laço "enquanto" (linhas 11 e 12 do Algoritmo 3.1) é $O(i^2)$.

Algoritmo 3.1 $OTA(n, I, X)$

```
1  l ← 1;
2  C^{π1}_{x1} ← max(P_{x1}, E_{x1});
3  B_1 ← {x_1};
4  para i = 2, 3, ..., n − 1, n faça
5      se C^{π_{i−1}}_{x_{i−1}} + S_{(x_{i−1})(x_i)} + P_{x_i} < E_{x_i} então
6          l ← l + 1;
7          B_l ← {x_i};
8          C^{π_i}_{x_i} ← E_{x_i};
       senão
9          C^{π_i}_{x_i} ← C^{π_{i−1}}_{x_{i−1}} + S_{(x_{i−1})(x_i)} + P_{x_i};
10         B_l ← B_l ∪ {x_i};
11         enquanto B_l não estiver no seu ponto de mínimo e C^{π_i}_{x_1} > P_{x_1} faça
12             DeslocaBloco(B_l);
13             se B_l se unir ao bloco B_{l−1} então
14                 l ← l − 1;
               fim
           fim
       fim
   fim
```

Seja κ_i o número de vezes que a condição do laço "enquanto" do Algoritmo 3.1 é verificada na iteração $i \in \{2, 3, \ldots, n\}$ desse algoritmo. Dado que a complexidade do procedimento $DeslocaBloco(B_l)$ é $O(|B_l|) = O(i)$, a complexidade computacional do OTA é:

$$O\left(\sum_{i=2}^{n}(i^2 + i) \cdot (\kappa_i - 1)\right) = O\left(\sum_{i=2}^{n} i^2 \cdot \kappa_i\right).$$

Lema 3.2 *Se κ_i representa o número de vezes que a condição do laço "enquanto" (linha 11) do Algoritmo 3.1 é verificada na iteração $i \in \{2, 3, \ldots, n\}$ desse algoritmo, então*

$$\sum_{i=2}^{n} \kappa_i \leq 2 \cdot (n - 1).$$

Prova: Verificar a condição do laço "enquanto" consiste, basicamente, em determinar o ponto de mínimo do bloco B_l. O ponto de mínimo de B_l é determinado somente após uma tarefa x_i ser alocada no final do bloco B_l corrente (linha 10 do Algoritmo 3.1) ou após o bloco B_l ser deslocado pelo procedimento $DeslocaBloco(B_l)$ (linha 12 do Algoritmo 3.1). O maior número possível de alocações de tarefas no final do bloco corrente é $n - 1$. Por outro lado, quando o bloco B_l é deslocado pelo procedimento $DeslocaBloco(B_l)$, ocorre uma das três seguintes situações:

(i) $C^{\pi_i}_{x_1} = P_{x_1}$;

(ii) B_l é programado no seu ponto de mínimo; (ou até que);

(iii) B_l é deslocado até se unir ao bloco B_{l-1}, dando origem a um novo bloco B_{l-1}.

Se ocorrer os casos (i) ou (ii), não será mais necessário determinar o ponto de mínimo do bloco B_l na iteração i. Por outro lado, caso aconteça o caso (iii), o ponto de mínimo do bloco B_l deve ser determinado novamente até que ocorra o caso (i) ou o caso (ii). Dado que, uma vez que

dois blocos se unem, eles não se separam mais, o número máximo de vezes que ocorre a união de blocos é, também, $n-1$. Logo, $2 \cdot (n-1)$ é um limite superior para $\sum_{i=2}^{n} \kappa_i$. □

Como consequência do Lema 3.2, tem-se:

$$\begin{aligned}
\sum_{i=2}^{n} i^2 \cdot \kappa_i &= 2^2 \cdot \kappa_2 + 3^2 \cdot \kappa_3 + 4^2 \cdot \kappa_4 + \ldots + (n-1)^2 \cdot \kappa_{n-1} + n^2 \cdot \kappa_n = \\
&= 2^2 \cdot (\underbrace{\kappa_2 + \kappa_3 + \kappa_4 + \ldots + \kappa_{n-1} + \kappa_n}_{\leq 2\cdot(n-1)}) + \\
&\quad + (3^2 - 2^2) \cdot (\underbrace{\kappa_3 + \kappa_4 + \ldots + \kappa_{n-1} + \kappa_n}_{\leq 2\cdot(n-1)}) + \\
&\quad + (4^2 - 3^2) \cdot (\underbrace{+\kappa_4 + \ldots + \kappa_{n-1} + \kappa_n}_{\leq 2\cdot(n-1)}) + \\
&\quad \vdots \\
&\quad + \left(n^2 - (n-1)^2\right) \cdot (\underbrace{\kappa_n}_{\leq 2\cdot(n-1)}).
\end{aligned}$$

Logo,

$$\sum_{i=2}^{n} i^2 \cdot \kappa_i \leq n^2 \cdot 2 \cdot (n-1) = 2n^3 - 2n^2.$$

Os resultados apresentados demonstram a seguinte proposição:

Proposição 3.3 *A complexidade computacional do OTA de Wan & Yen (2002) é* $O(n^3)$.

3.1.2 Novo algoritmo de alocação ótima de tempos ociosos – AAOTO

Nesta seção é proposto um algoritmo $O(n^2)$ para a alocação ótima de tempos ociosos em uma dada sequência de execução das tarefas do SMSPETP-SDS, denominado AAOTO. Esse algoritmo foi motivado pelo trabalho de França Filho (2007), que aborda o problema de programação de tarefas em máquinas paralelas não-relacionadas, tempos de preparação da máquina dependentes da sequência de execução das tarefas, datas de conclusão distintas, penalidades por antecipação e atraso da produção e datas distintas de liberação das tarefas.

Seja $X = (x_1, x_2, \ldots, x_n)$ uma dada sequência das tarefas pertencentes ao conjunto I. Logo, $x_i, x_j \in I$ e $x_i \neq x_j$ para todo $i, j \in \{1, 2, \cdots, n\}$ e $i \neq j$. O AAOTO é constituído de duas etapas. Primeiro, todas as tarefas são programadas para serem executadas o mais cedo possível, respeitando a sequência X. Observe que não haverá ociosidade de máquina nessa programação inicial. Além disso, os custos por antecipações e por atrasos são, respectivamente, os maiores e os menores possíveis para a sequência X. Em seguida, são alocados tempos ociosos entre cada par de tarefas consecutivas, sempre que a soma total das penalidades por antecipações e atrasos for reduzida.

Seja C_x^π a data de conclusão da tarefa $x \in I$ na programação π da sequência X. Na primeira etapa do AAOTO, a execução da primeira tarefa da sequência X, x_1, é programada para ser iniciada na data 0 (zero), ou seja, $C_{x_1}^\pi = P_{x_1}$. A data de conclusão das demais tarefas é dada por $C_{x_i}^\pi = C_{x_{i-1}}^\pi + S_{(x_{i-1})(x_i)} + P_{x_i}$, para todo $i = 2, 3, \ldots, n$.

Diferentemente do algoritmo de França Filho (2007), a segunda etapa do AAOTO começa a partir da última tarefa da sequência X (x_n). Sucessivamente, da última para a primeira tarefa, verifica-se se a alocação de tempos ociosos é vantajosa. Se a tarefa x_n estiver antecipada na programação resultante da primeira etapa do AAOTO (isto é, se $C_{x_n}^\pi < E_{x_n}$), então ela tem a sua execução adiada por $\varphi = T_{x_n} - C_{x_n}^\pi$ unidades de tempo. Caso contrário (isto é, se

$C^{\pi}_{x_n} \geq E_{x_n}$), a programação de x_n não é alterada, uma vez que adiar a sua execução não irá reduzir as penalidades associadas a ela.

Novamente, o conjunto de tarefas $B = \{x_u, x_{u+1}, \ldots, x_v\}$, com $u \leq v$, forma um bloco na programação π da sequência X se as tarefas em B são programadas consecutivamente sem tempo ocioso entre elas e existe ociosidade de máquina entre as tarefas x_{u-1} e x_u e as tarefas x_v e x_{v+1}. Para o caso em que $u = 1$, desconsidera-se a condição de existência de ociosidade entre as tarefas x_{u-1} e x_u. Analogamente, para o caso em que $v = n$, desconsidera-se a condição de existência de ociosidade entra as tarefas x_v e x_{v+1}.

A alocação de tempos ociosos antes das demais tarefas x_i, com $i = n-1, n-2, \ldots, 1$, é realizada da seguinte forma:

- Se $C^{\pi}_{x_i} \geq E_{x_i}$, então nenhum tempo ocioso é alocado imediatamente antes de x_i;
- Se $C^{\pi}_{x_i} < E_{x_i}$, então verifica-se se é vantajoso adiar a tarefa x_i.

Para adiar a tarefa x_i, é necessário adiar, também, todas as tarefas do bloco $B = \{x_i, x_{i+1}, \ldots, ult_{x_i}\}$, em que ult_{x_i} representa a última tarefa do bloco que contém x_i. O custo marginal CM para adiar as tarefas de B por uma unidade de tempo é dado por:

$$CM(B) = \sum_{x \in B\,:\, C^{\pi}_x \geq T_x} \beta_x - \sum_{x \in B\,:\, C^{\pi}_x < E_x} \alpha_x. \tag{3.2}$$

Em consequência:

- Se $CM(B) \geq 0$, então não é vantajoso adiar as tarefas de B (pois o aumento dos custos por atrasos será maior que a redução dos custos por antecipações);
- Se $CM(B) < 0$, então é vantajoso adiar as tarefas de B.

Devido à natureza convexa e linear por partes da função custo de B (ver Proposição 3.1), o valor de $CM(B)$ só irá mudar se as tarefas de B forem adiadas por, pelo menos:

$$\varphi = \min\left(C^{\pi}_{ult_{x_i}+1} - P_{ult_{x_i}+1} - S_{(ult_{x_i})(ult_{x_i}+1)} - C^{\pi}_{ult_{x_i}}, m_1, m_2\right)$$

unidades de tempo, sendo:

$$m_1 = \min_{x \in B\,:\, C^{\pi}_x < T_x} (T_x - C^{\pi}_x)$$

$$m_2 = \min_{x \in B \setminus \{ult_{x_i}\}\,:\, C^{\pi}_x < E_x} (E_x - C^{\pi}_x)$$

e $ult_{x_i} + 1$ representa a tarefa sequenciada imediatamente após a tarefa ult_{x_i}. Para o caso em que $ult_{x_i} = x_n$, o valor de φ é dado por $\varphi = \min(m_1, m_2)$. Portanto, as tarefas de B são adiadas por φ unidades de tempo. Consequentemente, os seguintes casos podem ocorrer:

- Se $C^{\pi}_{ult_{x_i}} + S_{(ult_{x_i})(ult_{x_i}+1)} + P_{ult_{x_i}+1} = C^{\pi}_{ult_{x_i}+1}$, então o bloco B é anexado ao bloco sucessor, formando um novo bloco B. Neste caso, deve-se verificar se é vantajoso adiar as tarefas do novo bloco B (via análise de $CM(B)$);
- Se $C^{\pi}_{ult_{x_i}} + S_{(ult_{x_i})(ult_{x_i}+1)} + P_{ult_{x_i}+1} < C^{\pi}_{ult_{x_i}+1}$, então verifica-se se é vantajoso adiar ainda mais as tarefas de B (via análise de $CM(B)$).

O AAOTO é encerrado quando não for mais vantajoso alocar ociosidade de máquina na programação π, ou seja, quando $CM(B) \geq 0$ para todo bloco B em π. A Proposição 3.2 garante que a programação obtida com o algoritmo AAOTO é ótima.

Tabela 3.2: Problema-teste para exemplificar o procedimento AAOTO.

I	P_x	E_x	T_x	α_x	β_x	S_{xy}			
						1	2	3	4
1	3	14	15	2	4	0	2	1	2
2	4	22	24	7	9	1	0	2	3
3	4	9	12	7	8	1	3	0	1
4	3	5	7	1	4	1	2	2	0

Para ilustrar o procedimento AAOTO, considere o SMSPETP-SDS de 4 tarefas apresentado novamente na Tabela 3.2. Nessa tabela, a primeira coluna apresenta o conjunto de tarefas a serem programadas, isto é, $I = \{1, 2, 3, 4\}$. As demais colunas indicam o tempo de execução (P_x), o custo por unidade de antecipação (α_x), o custo por unidade de atraso (β_x), a data de início (E_x) e de término (T_x) da janela de conclusão, bem como os tempos de preparação da máquina relativos a cada tarefa $x \in I$.

Dada a sequência $X = (3, 4, 1, 2)$, primeiramente todas as tarefas são programadas para serem executadas o mais cedo possível, conforme a Figura 3.7. Nessa programação, a soma das penalidades oriundas de antecipações e atrasos é igual a 71 unidades.

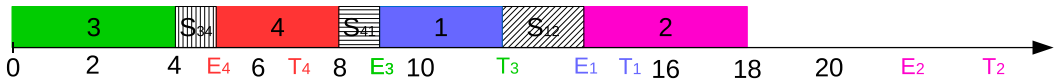


Figura 3.7: Programação inicial de X no AAOTO.

A segunda etapa do AAOTO verifica se é vantajoso alocar tempo ocioso antes da execução de cada tarefa. Esse procedimento é realizado iterativamente, começando pela última tarefa da sequência ($x_4 = 2$) e terminando com a primeira ($x_1 = 3$). Como a alocação de ociosidade de máquina antes da execução da tarefa $x_4 = 2$ reduz a sua antecipação ($CM(\{x_4\}) = -7 < 0$), a execução de x_4 é adiada o máximo possível sem que ela fique atrasada, ou seja, por 6 unidades de tempo (Figura 3.8). Após esse adiamento, a soma das penalidades por antecipações e atrasos é igual a 43.

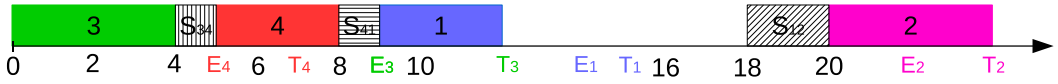


Figura 3.8: Programação de X após a iteração 1 da segunda etapa do AAOTO.

A iteração 2 da segunda etapa do AAOTO verifica se é vantajoso alocar ociosidade de máquina antes da execução da tarefa $x_3 = 1$. Como essa alocação de tempo ocioso é vantajosa, pois $CM(\{x_3\}) = -2 < 0$, a tarefa x_3 é adiada por 3 unidades de tempo, isto é, ela é adiada de modo a ser concluída no final da sua janela de conclusão. Após essa reprogramação (Figura 3.9), a soma das penalidades decorrentes de antecipações e atrasos é igual a 39 unidades.

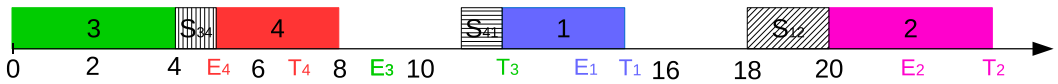


Figura 3.9: Programação de X após a iteração 2 da segunda etapa do AAOTO.

A iteração 3 da segunda etapa do AAOTO verifica se é vantajoso alocar ociosidade de máquina antes da execução da tarefa $x_2 = 4$. Como essa alocação de tempo ocioso não é

vantajosa ($CM(\{x_2\}) = 4 \geq 0$), o AAOTO vai para a iteração 4. A iteração 4 analisa se é vantajoso alocar tempo ocioso antes da execução da primeira tarefa da sequência ($x_1 = 3$). Para que essa alocação de tempo seja possível, é necessário adiar as tarefas do bloco $\{x_1, x_2\} = \{4, 3\}$. Já que esse adiamento é vantajoso ($CM(\{x_1, x_2\}) = -7+4 = -3 < 0$), o bloco $\{x_1, x_2\}$ é adiado por 3 unidades de tempo. Após esse adiamento, a soma das penalidades por antecipações e atrasos é igual a 30. Um novo bloco, $\{x_1, x_2, x_3\} = \{3, 4, 1\}$, é então formado (Figura 3.10). Nesse momento, é necessário verificar se esse novo bloco está na sua programação ótima. Uma vez que a alocação de mais ociosidade de máquina antes da primeira tarefa não é vantajosa ($CM(\{x_1, x_2, x_3\}) = -7 + 4 + 4 = 1 \geq 0$), tem-se que essa programação é ótima para a sequência X.

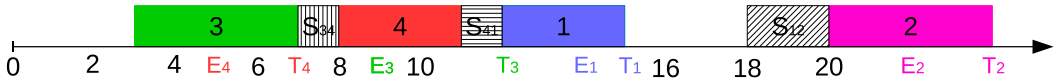


Figura 3.10: Programação ótima de X, obtida após a iteração 4 da segunda etapa do AAOTO.

3.1.2.1 Análise de complexidade do algoritmo AAOTO

O Algoritmo 3.2 apresenta o pseudocódigo do procedimento AAOTO aplicado à sequência $X = (x_1, x_2, \ldots, x_n)$ do conjunto I de n tarefas.

Algoritmo 3.2 $AAOTO(n, I, X)$

$C^{\pi}_{x_1} \leftarrow P_{x_1}$;
para $i = 2, \ldots, n$ **faça**
  $C^{\pi}_{x_i} \leftarrow C^{\pi}_{x_{i-1}} + S_{(x_{i-1})(x_i)} + P_{x_i}$;
fim
para $i = n, n-1, \ldots, 2, 1$ **faça**
  se $C^{\pi}_{x_i} < E_{x_i}$ **então**
    $B \leftarrow \{x_i, x_{i+1}, \ldots, ult_{x_i}\}$;
    enquanto $CM(B) < 0$ **faça**
      $m_1 \leftarrow \min_{x \in B\,:\, C^{\pi}_x < T_x} (T_x - C^{\pi}_x)$;
      $m_2 \leftarrow \min_{x \in B \setminus \{ult_{x_i}\}\,:\, C^{\pi}_x < E_x} (E_x - C^{\pi}_x)$;
      se $ult_{x_i} = x_n$ **então**
        $\varphi \leftarrow \min(m_1, m_2)$;
      senão
        $\varphi \leftarrow \min\left(C^{\pi}_{ult_{x_i}+1} - P_{ult_{x_i}+1} - S_{(ult_{x_i})(ult_{x_i}+1)} - C^{\pi}_{ult_{x_i}}, m_1, m_2\right)$;
      fim
      para $x \in B$ **faça**
        $C^{\pi}_x = C^{\pi}_x + \varphi$;
      fim
      se $ult_{x_i} \neq x_n$ *e* $C^{\pi}_{ult_{x_i}} + S_{(ult_{x_i})(ult_{x_i}+1)} + P_{ult_{x_i}+1} = C^{\pi}_{ult_{x_i}+1}$ **então**
        B é anexado ao bloco sucessor e forma um novo bloco B;
      fim
    fim
  fim
fim

A inicialização do AAOTO, linhas 1–3 do Algoritmo 3.2, possui complexidade computacional $O(n)$. Para cada $i \in \{n, n-1, \ldots, 2, 1\}$ do laço "para" da linha 4, o custo computacional de se verificar a condição da linha 5 é $O(1)$ e o custo associado à linha 6 é $O(|B|)$, em que $|B|$ corresponde à cardinalidade do conjunto B. A maior cardinalidade possível de B na iteração i é $n-i+1$ (ou seja, $|B| \leq n-i+1$, $\forall i \in \{n, n-1, \ldots, 2, 1\}$). Logo, a complexidade computacional relativa à linha 6 do Algoritmo 3.2 é $O(n-i+1)$.

O custo computacional de se verificar a condição do laço "enquanto" da linha 7 do Algoritmo 3.2, isto é, o custo de se calcular $CM(B)$, é $O(|B|) = O(n-i-1)$. Além disso, a complexidade computacional associada às linhas 8–16 também é $O(n-i+1)$. Assim, se cm_i representa o número de vezes que o laço "enquanto" (linhas 7-16 do Algoritmo 3.2) é executado na iteração $i \in \{n, n-1, \ldots, 2, 1\}$ desse algoritmo (isto é, cm_i corresponde ao número de vezes que $CM(B) < 0$ na iteração i), então a complexidade computacional do AAOTO é:

$$O\left(n + \sum_{i=1}^{n}\left((n-i+1) + \sum_{j=1}^{cm_i}(n-i+1)\right)\right) = O\left(n^2 + \sum_{i=1}^{n}(n-i+1)\cdot cm_i\right).$$

Lema 3.3 *Se cm_i representa o número de vezes que o laço "enquanto", linhas 7-16 do Algoritmo 3.2, é executado na iteração $i \in \{n, n-1, \ldots, 2, 1\}$ do AAOTO, então*

$$\sum_{i=1}^{n} cm_i \leq 3n - 1.$$

Prova: Toda vez que $CM(B) < 0$, as tarefas do bloco B são adiadas a fim de se obter a programação ótima desse bloco. Logo, existem duas possibilidades para cada vez que $CM(B) < 0$:

(i) O bloco B é adiado até que uma das tarefas em B seja concluída em um dos limites da sua respectiva janela de conclusão; ou

(ii) O bloco B é adiado até que ele seja anexado ao bloco sucessor, dando origem a um novo bloco B.

Cada limite da janela de conclusão de uma dada tarefa em I irá limitar o adiamento de um bloco por, no máximo, uma vez (visto que o AAOTO somente aloca ociosidade de máquina na programação de X). Assim, $2n$ é um limite superior para o número de vezes que ocorre o caso (i), pois existem n tarefas cujas janelas de conclusão possuem dois limites, E_x e T_x. Por outro lado, uma vez que dois blocos se juntam, eles não se separam mais. Portanto, $n-1$ é um limite superior para o número de vezes que ocorre o caso (ii) e $2n + (n-1) = 3n - 1$ é um limite superior para $\sum_{i=1}^{n} cm_i$. □

Como consequência do Lema 3.3, tem-se:

$$\begin{aligned}
\sum_{i=1}^{n}(n-i+1)\cdot cm_i &= n\cdot cm_1 + (n-1)\cdot cm_2 + \ldots + 2\cdot cm_{n-1} + 1\cdot cm_n \\
&= \underbrace{cm_1 + cm_2 + \ldots + cm_{n-1} + cm_n}_{\leq 3\cdot n-1} + \\
&\quad + \underbrace{cm_1 + cm_2 + \ldots + cm_{n-1}}_{\leq 3\cdot(n-1)-1} + \\
&\quad\quad \vdots \\
&\quad + \underbrace{cm_1 + cm_2}_{\leq 3\cdot 2-1} + \\
&\quad + \underbrace{cm_1}_{\leq 3\cdot 1-1}.
\end{aligned}$$

Logo,

$$\sum_{i=1}^{n}(n-i+1)\cdot cm_i \leq \sum_{i=1}^{n} 3i-1 = \frac{n(3n+1)}{2}.$$

A seguinte proposição resume os resultados apresentados.

Proposição 3.4 *A complexidade computacional do AAOTO é* $O(n^2)$.

3.1.3 *Optimal Timing Algorithm* melhorado – OTA'

Nesta seção é proposta uma alteração no algoritmo OTA, reproduzido na Seção 3.1.1 para o SMSPETP-SDS, a fim de reduzir a complexidade computacional desse algoritmo. O OTA com a alteração proposta será denotado por OTA'.

Seja $X = (x_1, x_2, \ldots, x_n)$ uma dada sequência das tarefas de I. Assim como no OTA, a primeira tarefa (x_1) da sequência X é programada para ser concluída na data E_{x_1}, se $P_{x_1} \leq E_{x_1}$, ou iniciada na data 0, se $P_{x_1} > E_{x_1}$. Seja C_x^π a data de conclusão da tarefa $x \in I$ na programação π da sequência X. Ainda da mesma forma que no OTA, as demais tarefas x_i, com $i \in \{2, 3, \ldots, n\}$, são programadas da seguinte forma:

(*i*) Se $C^\pi_{x_{i-1}} + S_{(x_{i-1})(x_i)} + P_{x_i} < E_{x_i}$, então a tarefa x_i é programada para ser finalizada na data E_{x_i} e um novo bloco é iniciado;

(*ii*) Se $C^\pi_{x_{i-1}} + S_{(x_{i-1})(x_i)} + P_{x_i} \geq E_{x_i}$, então a tarefa x_i é programada para ser concluída na data $C^\pi_{x_{i-1}} + S_{(x_{i-1})(x_i)} + P_{x_i}$ e passa a ser a última tarefa do bloco em que ela foi alocada, isto é, a tarefa x_i é programada para ser executada imediatamente após a conclusão da tarefa x_{i-1} e a realização da respectiva preparação da máquina.

Se acontecer o caso (*ii*), em que existe a possibilidade de uma tarefa ser programada para ser concluída com atraso, então deve-se verificar se é vantajoso adiantar a tarefa x_i. Para isso, seja $prim_{x_i}$ a primeira tarefa do bloco que contém a tarefa x_i. Para adiantar a tarefa x_i, é necessário adiantar, também, todas as tarefas do bloco $B = \{prim_{x_i}, prim_{x_i}+1, \ldots, x_i\}$. O custo marginal CM para adiantar as tarefas de B por uma unidade de tempo é dado por

$$CM(B) = \sum_{x\in B\,:\, C_x^\pi \leq E_x} \alpha_x - \sum_{x\in B\,:\, C_x^\pi > T_x} \beta_x. \tag{3.3}$$

Logo:

- Se $CM(B) \geq 0$, então não é vantajoso adiantar as tarefas de B (pois o aumento dos custos por antecipações será maior que a redução dos custos por atrasos);
- Se $CM(B) < 0$, então é vantajoso adiantar as tarefas de B.

Devido à natureza convexa e linear por partes da função custo de B (ver Proposição 3.1), o valor de $CM(B)$ só irá mudar se as tarefas de B forem adiantadas por, pelo menos,

$$\varphi = \min\left(C^\pi_{prim_{x_i}} - P_{prim_{x_i}} - S_{(prim_{x_i}-1)(prim_{x_i})} - C^\pi_{prim_{x_i}-1}, m_1, m_2\right)$$

unidades de tempo, sendo

$$m_1 = \min_{x\in B\,:\, C_x^\pi > T_x} (C_x^\pi - T_x)$$

$$m_2 = \min_{x\in B\,:\, C_x^\pi > E_x} (C_x^\pi - E_x)$$

e $prim_{x_i} - 1$ representa a tarefa sequenciada imediatamente antes da tarefa $prim_{x_i}$. Para o caso em que $prim_{x_i} = x_1$, o valor de φ é dado por $\varphi = \min(m_1, m_2)$. As tarefas de B são, então, adiantadas por φ unidades de tempo. Consequentemente, tem-se:

- Se $C^{\pi}_{prim_{x_i}-1} + S_{(prim_{x_i}-1)(prim_{x_i})} + P_{prim_{x_i}} = C^{\pi}_{prim_{x_i}}$, então o bloco B é anexado ao bloco antecessor, formando um novo bloco B. Neste caso, deve-se verificar se é vantajoso adiantar as tarefas do novo bloco B (via análise de $CM(B)$);

- Se $C^{\pi}_{prim_{x_i}-1} + S_{(prim_{x_i}-1)(prim_{x_i})} + P_{prim_{x_i}} < C^{\pi}_{prim_{x_i}}$, então verifica-se se é vantajoso adiantar ainda mais as tarefas de B (via análise de $CM(B)$).

  Observa-se que, diferentemente do OTA, que procura pelo ponto de mínimo do bloco B, o OTA' primeiro verifica se B está no seu ponto de mínimo (por meio da análise do $CM(B)$) e, caso não esteja, procura pelo ponto extremo de custo menor mais próximo do ponto atual.

O OTA' é encerrado quando não for mais vantajoso adiantar a execução das tarefas na programação π, ou seja, quando $CM(B) \geq 0$ para todo bloco B em π. Novamente, a Proposição 3.2 garante que a programação obtida com o algoritmo OTA' é ótima.

3.1.3.1 Análise de complexidade do algoritmo OTA'

O Algoritmo 3.3 descreve o procedimento OTA' aplicado a uma dada sequência de tarefas $X = \{x_1, x_2, \ldots, x_n\}$.

Algoritmo 3.3 $OTA^*(n, I, X)$

```
1  C^π_{x_1} ← max(P_{x_1}, E_{x_1});
2  para i = 2, 3, ..., n − 1, n faça
3      se C^π_{x_{i−1}} + S_{(x_{i−1})(x_i)} + P_{x_i} < E_{x_i} então
4          C^π_{x_i} ← E_{x_i};
       senão
5          C^π_{x_i} ← C^π_{x_{i−1}} + S_{(x_{i−1})(x_i)} + P_{x_i};
6          B ← {prim_{x_i}, prim_{x_i} + 1, ..., x_{i−1}, x_i};
7          enquanto CM(B) < 0  e  C^π_{prim_{x_i}} ≠ P_{prim_{x_i}} faça
8              m_1 ← min_{x∈B : C^π_x > T_x} (C^π_x − T_x);
9              m_2 ← min_{x∈B : C^π_x > E_x} (C^π_x − E_x);
10             se prim_{x_i} = x_1 então
11                 φ ← min(C^π_{x_1} − P_{x_1}, m_1, m_2);
               senão
12                 φ ← min(C^π_{prim_{x_i}} − P_{prim_{x_i}} − S_{(prim_{x_i}−1)(prim_{x_i})} − C^π_{prim_{x_i}−1}, m_1, m_2);
               fim
13             para x ∈ B faça
14                 C^π_x = C^π_x − φ;
               fim
15             se prim_{x_i} ≠ x_1  e  C^π_{prim_{x_i}} − P_{prim_{x_i}} − S_{(prim_{x_i}−1)(prim_{x_i})} = C^π_{prim_{x_i}−1} então
16                 B é anexado ao bloco antecessor e forma um novo bloco B;
               fim
           fim
       fim
   fim
```

O custo computacional associado à linha 1 do Algoritmo 3.3 é constante. Para cada iteração $i \in \{2, 3, \ldots, n\}$ do laço "para" da linha 2, a complexidade computacional de se verificar a

condição da linha 3 e a complexidade associada às linhas 4 e 5 também são constantes. O custo associado à linha 6 é $O(|B|)$, em que $|B|$ corresponde à cardinalidade do conjunto B. A maior cardinalidade possível de B na iteração i é i (ou seja, $|B| \leq i$, $\forall i \in \{2, 3, \ldots, n\}$). Portanto, a complexidade computacional relativo à linha 6 do Algoritmo 3.3 é $O(i)$. O custo computacional de se verificar a condição da linha 7 do Algoritmo 3.3, isto é, o custo de se calcular $CM(B)$, é $O(|B|) = O(i)$. Além disso, a complexidade computacional associada às linhas 8–16 também é $O(i)$. Seja cm_i o número de vezes que o laço "enquanto" (linhas 7-16 do Algoritmo 3.3) é executado na iteração $i \in \{2, 3, \ldots, n\}$ desse algoritmo (isto é, cm_i corresponde ao número de vezes que $CM(B) < 0$ na iteração i), então a complexidade computacional do OTA' é

$$O\left(1 + \sum_{i=2}^{n}\left((1+i) + \sum_{j=1}^{cm_i} i\right)\right) = O\left(n^2 + \sum_{i=2}^{n} i \cdot cm_i\right).$$

Lema 3.4 *Se cm_i representa o número de vezes que o laço "enquanto", linhas 7-16 do Algoritmo 3.3, é executado na iteração $i \in \{2, 3, \ldots, n\}$ do OTA', então*

$$\sum_{i=2}^{n} cm_i \leq 3n - 1.$$

Prova: Análoga a do Lema 3.3. □

Como consequência do Lema 3.4, tem-se:

$$\begin{aligned}
\sum_{i=2}^{n} i \cdot cm_i &= 2 \cdot cm_2 + 3 \cdot cm_3 + \ldots + (n-1) \cdot cm_{n-1} + n \cdot cm_n \\
&= \underbrace{cm_2 + cm_3 + \ldots + cm_{n-1} + cm_n}_{\leq 3n-1} + \\
&\quad + \underbrace{cm_2 + cm_3 + \ldots + cm_{n-1} + cm_n}_{\leq 3n-1} + \\
&\quad + \underbrace{cm_3 + \ldots + cm_{n-1} + cm_n}_{\leq 3n-1} + \\
&\quad \vdots \\
&\quad + \underbrace{cm_{n-1} + cm_n}_{\leq 3n-1} + \\
&\quad + \underbrace{cm_n}_{\leq 3n-1}.
\end{aligned}$$

Logo,

$$\sum_{i=2}^{n} i \cdot cm_i \leq n \cdot (3n - 1).$$

Os resultados apresentados demonstram a seguinte proposição:

Proposição 3.5 *A complexidade computacional do OTA' é $O(n^2)$.*

3.2 VNS aplicado ao problema

Nesta seção são propostos três algoritmos baseados na metaheurística *Variable Neighborhood Search* (VNS) para resolver o SMSPETP-SDS. A metaheurística VNS é descrita na Seção 3.2.1, ao passo que as soluções do SMSPETP-SDS por meio desta metaheurística são propostas na Seção 3.2.2.

3.2.1 Descrição do VNS

Busca em Vizinhança Variável (*Variable Neighborhood Search* – VNS) é uma metaheurística que explora sistematicamente a troca de estruturas de vizinhança, tanto para encontrar quanto para escapar de um ótimo local (Hansen *et al.*, 2008). O VNS foi escolhido para resolver o SMSPETP-SDS por ser uma metaheurística que possui poucos parâmetros para serem calibrados e que se mostrou eficiente na resolução de diversos problemas combinatórios (Hansen & Mladenović, 2014).

O VNS é baseado em três fatos básicos:

- Um ótimo local em relação a uma dada estrutura de vizinhança pode não ser um ótimo local em relação a outra vizinhança;
- Um ótimo global é um ótimo local em relação a todas as possíveis estruturas de vizinhança;
- Para muitos problemas, ótimos locais em relação a uma ou mais estruturas de vizinhança são relativamente próximos.

O último fato apresentado é de natureza empírica e sugere que um ótimo local fornece informações importantes sobre o ótimo global.

O método VNS explora estruturas de vizinhança gradativamente mais distantes da solução corrente, sendo que a solução corrente é atualizada se, e somente se, uma solução melhor for encontrada. Além disso, uma rotina de busca local é aplicada em cada solução vizinha explorada. Essa rotina de busca local também pode usar diferentes estruturas de vizinhança (Hansen & Mladenović, 2001).

O método VNS é apresentado no Algoritmo 3.4. Nele é considerado o problema de minimização de uma função $F(.)$. A solução inicial X e o conjunto $N = \{N_1, N_2, \ldots, N_r\}$ formado por r estruturas de vizinhança diferentes são dados de entrada do método.

Algoritmo 3.4 VNS($F(.)$, N, r, X)

```
enquanto Critério de parada não for satisfeito faça
    k ← 1;                                          /* Vizinhança corrente */
    enquanto k ≤ r faça
        Gere, aleatoriamente, uma solução vizinha X' ∈ N_k(X);
        X'' ← BuscaLocal(X');
        se F(X'') < F(X) então
            X ← X'';
            k ← 1;
        senão
            k ← k + 1;                              /* Troca da vizinhança */
        fim
    fim
fim
Retorne X;
```

Conforme o Algoritmo 3.4, o VNS parte de uma solução X e, em cada iteração do método, uma solução vizinha (X') em relação à vizinhança N_k de X é gerada aleatoriamente. Essa solução vizinha é então submetida a uma rotina de busca local. Se a solução retornada pela busca local (X'') for melhor que a solução corrente, então a busca recomeça a partir da solução X'', que passa a ser a solução corrente, e da primeira estrutura de vizinhança (N_1). Caso contrário, uma nova solução X' em relação à vizinhança N_{k+1} de X é gerada aleatoriamente.

O método VNS é encerrado quando um critério de parada for atingido, como, por exemplo, um número máximo de iterações consecutivas sem melhora na solução corrente.

Se a rotina de busca local do VNS for a heurística de Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND), apresentada em Mladenović & Hansen (1997), então o método recebe a denominação *General Variable Neighborhood Search* (GVNS).

Assim como o VNS, o VND também é baseado no fato de que um ótimo local em relação a uma dada estrutura de vizinhança pode não ser um ótimo local em relação a outra vizinhança (Hansen & Mladenović, 2005). O método VND clássico recebe como entrada uma solução inicial e um conjunto finito e ordenado de estruturas de vizinhança. Em cada iteração, uma estrutura de vizinhança é explorada até que se encontre um ótimo local em relação a ela. Se esse ótimo local representar uma melhora na solução corrente, ele passa a ser a solução corrente e retorna-se à primeira vizinhança. Caso contrário, passa-se a explorar a vizinhança seguinte. O método para quando um ótimo local em relação a todas as estruturas de vizinhança for encontrado.

O Algoritmo 3.5 representa a busca local VND básica. Nele é considerado o problema de minimização de uma função $F(.)$, dado uma solução inicial X e um conjunto $N = \{N_1, N_2, \ldots, N_r\}$ composto de r estruturas de vizinhança diferentes.

Algoritmo 3.5 *VND*($F(.)$, N, r, X)

```
k ← 1;                                              /* Vizinhança corrente */
enquanto (k ≤ r) faça
    Encontre o melhor vizinho X' ∈ N_k(X)
    se (F(X') < F(X)) então
        X ← X';
        k ← 1;
    senão
        k ← k + 1;                                  /* Troca da vizinhança */
    fim
fim
Retorne X
```

Segundo Souza (2011), dependendo do problema abordado, a busca pelo ótimo local (X') pode ser custosa computacionalmente. Nessa situação, é comum fazer a busca pela primeira solução de melhora. Outra alternativa é considerar a exploração de apenas um certo percentual da vizinhança, sendo esse percentual um parâmetro do método. Uma terceira alternativa, bastante utilizada nessas situações, é aplicar o método randômico de descida em cada vizinhança explorada. No método randômico de descida, iterativamente, um vizinho é escolhido de forma aleatória. Se esse vizinho escolhido for de melhora, então ele passa a ser a solução corrente. Caso contrário, a solução corrente permanece a mesma e passa-se para a próxima iteração. O método randômico de descida é interrompido após um número pré-determinado de iterações sem melhora na solução corrente.

Mais informações sobre o VNS, o VND e o GVNS podem ser encontradas em Mladenović & Hansen (1997); Hansen & Mladenović (2001); Hansen & Mladenović (2003); Hansen & Mladenović (2005); Hansen *et al.* (2008); Hansen & Mladenović (2014).

3.2.2 Solução do SMSPETP-SDS via GVNS

Esta seção propõe três algoritmos baseados na metaheurística *General Variable Neighborhood Search* (GVNS) para o SMSPETP-SDS, denominados:

(i) GVNS$_r$;

(ii) GVNS$_f$; e

(iii) GVNS$_{rf}$.

Todos eles usam o algoritmo GVNS clássico, o qual é apresentado no Algoritmo 3.6.

Algoritmo 3.6 GVNS$(I, n, F(.), N_1(.), N_2(.), N_3(.),$ GVNSmax$)$

```
X ← SoluçãoInicial(I, n);
f* ← F(X);
Iter ← 0;
enquanto Iter < GVNSmax faça
    Iter ← Iter + 1;
    k ← 1;
    enquanto k ≤ 3 faça
        Gere, aleatoriamente, um vizinho X' ∈ N_k(X);
        X'' ← VND(X', F(.), N_1(.), N_2(.), N_3(.), VNDmax);
        se F(X'') < F(X) então
            X ← X'';
            f* ← F(X);
            k ← 1;
            Iter ← 0;
        senão
            k ← k + 1;
        fim
    fim
fim
Retorne f*;
```

No Algoritmo 3.6, GVNSmax representa o número máximo de iterações sem melhora na melhor solução encontrada e define o critério de parada do algoritmo. Os detalhes dos procedimentos SoluçãoInicial, F e VNDs propostos, bem como as descrições das vizinhanças N_1, N_2 e N_3, são apresentados nas subseções seguintes.

3.2.2.1 Representação de uma solução

Uma solução (sequência) para o SMSPETP-SDS de n tarefas é representada por uma sequência X de n posições. Cada posição $i = 1,\ 2,\ \cdots,\ n$ indica a ordem de execução da tarefa x_i. Por exemplo, na sequência $X = (5,\ 1,\ 2,\ 6,\ 4,\ 3)$, a tarefa 5 é a primeira a ser executada e a tarefa 3 é a última.

3.2.2.2 Solução inicial

A solução inicial para o SMSPETP-SDS utilizada consiste em sequenciar as tarefas de acordo com a ordem crescente das datas de início de suas janelas de conclusão. Em caso de empate, sequencia-se primeiro a tarefa que possui a janela de conclusão com menor data de término. Se persistir o empate, sequencia-se primeiro a tarefa de menor índice.

3.2.2.3 Vizinhança de uma solução

Para explorar o espaço de soluções, são utilizados três tipos de movimentos:

(i) m_1: troca das posições de duas tarefas na sequência de execução;

(ii) m_2: realocação de uma tarefa para outra posição na sequência; e

(iii) m_3: realocação de uma subsequência de k tarefas, $1 \leq k < n$, para outra posição na sequência de execução.

Esses movimentos definem as três vizinhanças N_1, N_2 e N_3, respectivamente, descritas a seguir.

(a) Vizinhança N_1: Um exemplo de vizinho da solução $X = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$ na vizinhança N_1 é a solução $X' = (5,\ 4,\ 2,\ 1,\ 3,\ 6)$. Observa-se que X' é obtida de X por meio da troca das posições de execução das tarefas 3 e 4.

Para uma dada sequência de n tarefas, a posição de cada tarefa pode ser trocada com a posição de qualquer uma das demais $n-1$ tarefas. Por outro lado, trocar as posições da i-ésima com a j-ésima tarefa na sequência é equivalente a trocar de posições a j-ésima e a i-ésima tarefa na sequência, para todo $i, j \in \{1, 2, \cdots, n\}$. Portanto, há $n(n-1)/2$ vizinhos distintos em relação à vizinhança N_1.

(b) Vizinhança N_2: A solução $X' = (5,\ 2,\ 1,\ 4,\ 3,\ 6)$ é um exemplo de vizinho da solução $X = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$ em relação à vizinhança N_2. Nota-se que X' é obtida de X ao realocar a tarefa 3, que está na segunda posição em X, para a quinta posição na sequência de execução.

Dada uma sequência de n tarefas, cada uma delas pode ser realocada para $n-1$ posições distintas. Além disso, realocar a tarefa da i-ésima posição para a posição $i+1$ na sequência de execução é equivalente a realocar a tarefa da posição $i+1$ para a i-ésima posição na sequência, para todo $i \in \{1,\ 2,\ \cdots,\ n-1\}$. Portanto, há $(n-1)^2$ vizinhos distintos em relação à vizinhança N_2.

(c) Vizinhança N_3: Um vizinho da solução $X = (5,\ 3,\ 2,\ 1,\ 4,\ 6)$ na vizinhança N_3 é a solução $X' = (1,\ 4,\ 5,\ 3,\ 2,\ 6)$. X' é obtida de X ao realocar a subsequência de três tarefas consecutivas $(5,\ 3,\ 2)$, que estão nas três primeiras posições em X, para depois da tarefa 4.

Dada uma sequência com n tarefas, existem $n-k+1$ subsequências distintas de k tarefas consecutivas, para todo $k \in \{1,\ 2,\ \cdots,\ n\}$. Cada uma dessas subsequências pode ser realocada para $n-k$ posições distintas na sequência de execução. Ademais, realocar uma subsequência de k_1 tarefas consecutivas para k_2 posições sucessoras é equivalente a realocar uma subsequência de k_2 tarefas consecutivas para k_1 posições antecessoras, para todo $k_1,\ k_2 \in \{1,\ 2,\ \cdots,\ n\}$ e $k_1 + k_2 \leq n$. Logo, há $n\,(n-1)\,(n+1)\,/6$ vizinhos distintos em relação à vizinhança N_3.

Com o objetivo de evitar o retorno a vizinhos já analisados e, consequentemente, reduzir o custo computacional, todas as observações mencionadas anteriormente foram consideradas na implementação computacional dos algoritmos experimentados. Dado que uma solução foi implementada como um vetor nos experimentos computacionais, a ordem escolhida para a exploração das vizinhanças explora vizinhanças cada vez mais distantes da solução corrente em termos de complexidade computacional, conforme proposto por Hansen *et al.* (2008).

3.2.2.4 Avaliação de uma solução

Qualquer sequência de tarefas é uma solução factível para o SMSPETP-SDS. A fim de avaliar uma sequência $X = (x_1, x_2, \ldots, x_n)$, primeiro aplica-se em X um dos algoritmos de programação ótima de uma dada sequência de tarefas apresentados na Seção 3.1. Então, utiliza-se a programação ótima de X, dada por π^*_X, para calcular o valor da função objetivo de X,

dada por:

$$F(X) = f(\pi_X^*) = \sum_{i=1}^{n} \alpha_{x_i} \cdot \max\left(0, E_{x_i} - C_{x_i}^{\pi_X^*}\right) + \beta_{x_i} \cdot \max\left(0, C_{x_i}^{\pi_X^*} - T_{x_i}\right) \qquad (3.4)$$

e que deve ser minimizada.

3.2.2.5 VNDs utilizados

São utilizadas duas variações do método de busca local VND (Mladenović & Hansen, 1997), denominadas VND_r e VND_f, descritas a seguir.

(a) VND_r: A busca local VND_r utiliza a seguinte sequência de buscas locais:

(ii) BL1: Descida randômica na vizinhança N_1;

(ii) BL2: Descida randômica na vizinhança N_2; e

(iii) BL3: Descida randômica na vizinhança N_3.

Na primeira busca local (BL1), duas tarefas são escolhidas aleatoriamente e suas posições na sequência de execução são trocadas. Se a nova sequência de execução for uma solução melhor, então ela é aceita e passa a ser a solução corrente. Caso contrário, é testada outra troca aleatória na solução corrente. A BL1 termina quando ocorrer VNDmax tentativas de trocas consecutivas sem melhora na solução corrente, sendo VNDmax um parâmetro do procedimento. Quando a BL1 termina, aplica-se a próxima busca local (BL2).

Na busca local BL2, aleatoriamente escolhe-se uma tarefa na sequência de execução e uma nova posição para ela. Se a nova sequência for uma solução melhor, então ela passa a ser a solução corrente e o VND volta para a busca local BL1. Caso contrário, é testada outra realocação aleatória na solução corrente. A BL2 é interrompida após VNDmax tentativas de realocações consecutivas sem melhora na solução corrente. Nesse último caso, passa-se para a busca local seguinte (BL3).

Na busca local BL3, aleatoriamente escolhe-se uma subsequência de tarefas na sequência de execução e uma nova posição para ela. O tamanho da subsequência também é escolhido de forma aleatória no intervalo $[1,\ n\ -\ 1\ ]$. Se a nova sequência de execução for uma solução melhor, então ela passa a ser a solução corrente e o VND volta para a busca local BL1. Caso contrário, é testada outra realocação de subsequência aleatória na solução corrente. A BL3 é interrompida após VNDmax tentativas de realocações de subsequências consecutivas sem melhora na solução corrente. Nesse último caso, o VND é encerrado e a melhor solução encontrada é retornada.

É interessante observar que a solução resultante do VND_r não é necessariamente um ótimo local em relação às vizinhanças adotadas, visto que as respectivas vizinhanças não são, de um modo geral, completamente exploradas em cada busca local.

(b) VND_f: A busca local VND_f utiliza a seguinte sequência de buscas locais:

(i) BL4: Descida Primeiro de Melhora com a vizinhança N_1; e

(ii) BL5: Descida Primeiro de Melhora com a vizinhança N_3.

Quando não houver soluções vizinhas que melhorem a solução corrente X durante a busca local BL4, passa-se para a busca local BL5. A busca local BL5 inicia-se testando as realocações de subsequências formadas por apenas uma tarefa. Quando não for possível

melhorar a solução corrente com essas realocações, o tamanho das subsequências realocadas é incrementado em uma unidade, isto é, são testadas as realocações de subsequências formadas por 2 tarefas. Sempre que não for possível melhorar a solução corrente com realocações de subsequências de um dado tamanho, passa-se a testar realocações de subsequências de tamanho imediatamente superior, até que o tamanho das subsequências realocadas seja igual a $n-1$. Sempre que uma solução vizinha X' que melhora a solução corrente X é encontrada, X' passa a ser a nova solução corrente e volta-se para a busca local BL4. O procedimento VND_f é encerrado quando a busca local BL5 testar todas as possíveis realocações de subsequências de tarefas e não encontrar uma solução melhor que a solução oriunda da busca local BL4. Neste caso, a solução retornada é um ótimo local em relação às vizinhanças N_1 e N_3. Dado que a vizinhança N_2 corresponde a realocar subsequências de apenas uma tarefa, tem-se que a vizinhança N_2 está contida na vizinhança N_3. Logo, um ótimo local em relação à vizinhança N_3 é, também, um ótimo local em relação à vizinhança N_2.

3.2.2.6 GVNS_r, GVNS_f e GVNS_{rf}

O algoritmo GVNS_r é obtido ao se utilizar o procedimento VND_r como busca local VND do algoritmo GVNS (ver Algoritmo 3.6), enquanto o algoritmo GVNS_f é obtido ao se utilizar o procedimento VND_f como busca local. Visto que a solução retornada pelo procedimento VND_r não é necessariamente um ótimo local em relação às vizinhanças adotadas, propõe-se, também, o algoritmo GVNS_{rf}. Esse algoritmo consiste em aplicar a busca local VND_f na solução obtida pelo GVNS_r. Logo, a solução retornada pelo algoritmo GVNS_{rf} também corresponde a um ótimo local em relação às vizinhanças exploradas.

3.3 Enumeração implícita aplicada ao SMSPETP-SIS

Algoritmos de enumeração implícita são métodos determinísticos que fazem uso de certa "sabedoria" para encontrar a solução ótima de um problema de otimização, mesmo enumerando apenas parte do conjunto de soluções possíveis. A seguir é proposto um algoritmo de enumeração implícita, doravante denotado por EI, para resolver o SMSPETP-SIS. De acordo com Janiak *et al.* (2015), não existe na literatura um algoritmo de otimização exato para a solução do SMSPETP-SIS.

Para auxiliar na apresentação do algoritmo EI, seja I o conjunto de n tarefas que devem ser programadas. Seja UB um limite superior conhecido para o problema (obtido de forma heurística, por exemplo). Durante o processo de enumeração, um nó δ corresponde a uma estrutura que armazena uma subsequência de k tarefas ($0 \leq k \leq n$), bem como o conjunto das $n-k$ tarefas que estão fora dessa subsequência, representados por $seq(\delta)$ e $nseq(\delta)$, respectivamente. Seja L a lista de nós a serem investigados. Assim como na Seção 3.1, uma programação de uma sequência X das tarefas de I é uma programação de I que preserva a sequência de execução dada por X. A sequência X é avaliada tal como na Seção 3.2.2.4. Logo, a sequência X é ótima se, e somente se, existe uma programação ótima de I que respeita a sequência X.

Seja δ_0 o nó tal que $seq(\delta_0) = \varnothing$ e $nseq(\delta_0) = I$. O procedimento EI é inicializado com a inserção do nó δ_0 como o único nó de L. Depois disso, cada iteração de EI consiste nos seguintes passos:

Passo 1: Seja δ o último nó inserido em L;

Passo 2: Retira-se de L o nó δ;

Passo 3: Se $nseq(\delta) = \varnothing$, então a subsequência $seq(\delta)$ corresponde a uma solução completa. Além disso, caso $F(seq(\delta)) < UB$, então o valor de UB é atualizado para $UB = F(seq(\delta))$.

Passo 4: Se $nseq(\delta) \neq \varnothing$, então, para cada tarefa $x \in nseq(\delta)$, verifica-se se a subsequência $seq(\delta) \cup \{x\}$ obtida ao inserir a tarefa x no final da subsequência $seq(\delta)$ viola as condições de otimalidade (apresentadas na Seção 3.3.1). Caso essas condições não sejam violadas, o nó δ_{filho} tal que $seq(\delta_{filho}) = seq(\delta) \cup (x)$ e $nseq(\delta_{filho}) = nseq(\delta) \setminus \{x\}$ é inserido em L.

Como pode ser observado na descrição do algoritmo EI, a fim de se reduzir a quantidade de memória computacional utilizada, nesse algoritmo, a exploração da árvore de soluções é realizada por meio de uma busca em profundidade. O algoritmo EI é interrompido quando $L = \varnothing$ e o valor final de UB corresponde à solução ótima do problema.

3.3.1 Condições de otimalidade

Para facilitar a notação, seja $seq(\delta) = A = (x_1, x_2, \cdots, x_u)$, com $1 \leq u \leq n$, uma subsequência das tarefas de I. Além disso, seja X_A uma sequência das tarefas em I que inicia com a subsequência A.

Proposição 3.6 *Se $F(A) > UB$, então não existe uma sequência ótima de I que contém a subsequência A.*

Prova: Este resultado é direto, uma vez que a inclusão de tarefas em A não reduz a soma das penalidades das tarefas em programações de A. □

Lema 3.5 *Seja $x \in A$. Se $C_x^{\pi_A^*}$ representa a data de conclusão da tarefa x em uma programação ótima da subsequência A (π_A^*), então existe uma programação ótima de X_A ($\pi_{X_A}^*$) em que x é concluída na data $C_x^{\pi_{X_A}^*} \leq C_x^{\pi_A^*}$.*

Prova: Seja π'_{X_A} uma programação ótima de X_A. Dado que a tarefa x é concluída na data $C_x^{\pi_A^*}$ em π_A^*, se $C_x^{\pi'_{X_A}} > C_x^{\pi_A^*}$, então é possível adiantar a execução das tarefas em A na programação π'_{X_A} de modo que a tarefa x seja concluída na data $C_x^{\pi_A^*}$, sem aumentar a soma das penalidades das tarefas em A. Como esse adiantamento das tarefas em A não altera as penalidades das tarefas fora de A, tem-se que a programação resultante é a programação ótima $\pi_{X_A}^*$ procurada. □

Corolário 3.1 *Se existe uma programação ótima de A (π_A^*) na qual todas as tarefas são executadas consecutivamente, sem ociosidade de máquina entre elas e a execução da primeira é iniciada na data 0, então existe uma programação ótima de X_A ($\pi_{X_A}^*$) em que $C_x^{\pi_{X_A}^*} = C_x^{\pi_A^*}$, sendo $C_x^{\pi_{X_A}^*}$ e $C_x^{\pi_A^*}$ as datas de conclusão de x em $\pi_{X_A}^*$ e π_A^*, respectivamente.*

Prova Neste caso, $C_x^{\pi_A^*}$ é igual à soma dos tempos de execução das tarefas programadas antes de x. Portanto, a demonstração difere da prova do Lema 3.5 apenas pelo fato de que agora não é possível adiantar a execução de x. □

Proposição 3.7 *Se existem duas tarefas consecutivas x e y em A tais que $\alpha_y P_x < \alpha_x P_y$ e $C_x^{\pi_A^*} + P_y = C_y^{\pi_A^*} \leq \min(E_x, E_y)$ em uma programação ótima de A (π_A^*), então não existe uma sequência ótima de I que inicia com a subsequência A.*

Prova: Suponha que exista uma sequência ótima de I que inicia com A, dada por X_A. De acordo com o Lema 3.5, existe uma programação ótima de X_A, dada por $\pi^*_{X_A}$, tal que $C_y^{\pi^*_{X_A}} \leq C_y^{\pi^*_A} \leq \min(E_x, E_y)$. Logo, a soma das penalidades associadas às tarefas x e y em $\pi^*_{X_A}$ é dada por

$$\alpha_x \cdot (E_x - C_x^{\pi^*_{X_A}}) + \alpha_y \cdot (E_y - C_y^{\pi^*_{X_A}}),$$

ou seja,

$$\alpha_x \cdot (E_x - C_y^{\pi^*_{X_A}} + P_y) + \alpha_y \cdot (E_y - C_y^{\pi^*_{X_A}}). \tag{3.5}$$

Seja $X_{A'}$ uma sequência obtida de X_A por meio da troca da ordem de execução das tarefas x e y. Então, existe uma programação ótima de $X_{A'}$ ($\pi^*_{X_{A'}}$) em que a tarefa x é concluída na mesma data em que a tarefa y é concluída em $\pi^*_{X_A}$ (ou seja, $C_x^{\pi^*_{X_{A'}}} = C_y^{\pi^*_{X_A}}$), a tarefa y é concluída imediatamente antes do início da tarefa x (ou seja, $C_y^{\pi^*_{X_{A'}}} = C_x^{\pi^*_{X_{A'}}} - P_x$) e todas as demais tarefas são concluídas nas respectivas datas de conclusão em $\pi^*_{X_A}$ (isto é, $C_z^{\pi^*_{X_{A'}}} = C_z^{\pi^*_{X_A}}$, $\forall z \in I \setminus \{x, y\}$). Assim, a soma das penalidades associadas às tarefas x e y em $\pi^*_{X_{A'}}$ é dada por

$$\alpha_x \cdot (E_x - C_x^{\pi^*_{X_{A'}}}) + \alpha_y \cdot (E_y - C_y^{\pi^*_{X_{A'}}}),$$

isto é,

$$\alpha_x \cdot (E_x - C_y^{\pi^*_{X_A}}) + \alpha_y \cdot (E_y - C_y^{\pi^*_{X_A}} + P_x). \tag{3.6}$$

Como as datas de conclusão das tarefas diferentes de x e y são as mesmas nas programações $\pi^*_{X_A}$ e $\pi^*_{X_{A'}}$, segue que

$$f(\pi^*_{X_A}) - f(\pi^*_{X_{A'}}) = (3.5) - (3.6) = \alpha_x P_y - \alpha_y P_x > 0.$$

Isto contradiz o fato de X_A ser uma sequência ótima de I. □

Proposição 3.8 *Se existe uma programação ótima de A (π^*_A) na qual todas as tarefas são executadas consecutivamente sem ociosidade de máquina entre elas, a primeira tarefa é iniciada na data 0 e existem duas tarefas adjacentes x e y tais que $custo_1 > custo_2$, em que*

$$\begin{aligned} custo_1 &= \alpha_x \cdot \max(0, E_x - C_x^{\pi^*_A}) + \beta_x \cdot \max(0, C_x^{\pi^*_A} - T_x) + \\ &\quad + \alpha_y \cdot \max(0, E_y - C_y^{\pi^*_A}) + \beta_y \cdot \max(0, C_y^{\pi^*_A} - T_y) \quad e \\ custo_2 &= \alpha_x \cdot \max(0, E_x - C_x^{\pi^*_A} - P_y) + \beta_x \cdot \max(0, C_x^{\pi^*_A} + P_y - T_x) + \\ &\quad + \alpha_y \cdot \max(0, E_y - C_y^{\pi^*_A} + P_x) + \beta_y \cdot \max(0, C_y^{\pi^*_A} - P_x - T_y), \end{aligned}$$

então não existe uma programação ótima de I que inicia com a subsequência A.

Prova: Suponha que exista uma sequência ótima de I que inicia com A, X_A. Com base no Corolário3.1, existe uma programação ótima de X_A, $\pi^*_{X_A}$, tal que $C_x^{\pi^*_{X_A}} = C_x^{\pi^*_A}$ e $C_y^{\pi^*_{X_A}} = C_y^{\pi^*_A}$. Consequentemente, a soma das penalidades associadas às tarefas x e y é igual $custo_1$. Seja $X_{A'}$ uma sequência obtida de X_A por meio da troca da ordem de execução das tarefas x e y. Logo, existe uma programação de $X_{A'}$ ($\pi_{X_{A'}}$) em que $C_z^{\pi_{X_{A'}}} = C_z^{\pi^*_{X_A}}$, $\forall z \in I \setminus \{x, y\}$. A soma das penalidades associadas às tarefas x e y na programação $\pi_{X_{A'}}$ é dada por $custo_2$. Portanto, $f(\pi^*_{X_A}) - f(\pi_{X_{A'}}) = custo_1 - custo_2 > 0$. Isto contradiz o fato de X_A ser uma sequência ótima de I. □

Com base nas Proposições 3.6, 3.7 e 3.8, o seguinte corolário estabelece os critérios utilizados para verificar se uma subsequência $A = seq(\delta)$ viola as condições de otimalidade.

Corolário 3.2 *A subsequência $seq(\delta)$ viola as condições de otimalidade se ocorrer um dos seguintes casos:*

(i) $F(seq(\delta)) > UB$;

(ii) Existem duas tarefas consecutivas x e y em $seq(\delta)$ tais que $\alpha_y P_x < \alpha_x P_y$ *e* $C_x^{\pi^*_{seq(\delta)}} + P_y = C_y^{\pi^*_{seq(\delta)}} \leq \min(E_x, E_y)$, *em que $\pi^*_{seq(\delta)}$ representa uma programação ótima de $seq(\delta)$; ou*

(iii) Existe uma programação ótima de $seq(\delta)$ na qual todas as tarefas são executadas consecutivamente sem ociosidade de máquina entre elas, a primeira tarefa é iniciada na data 0 e existem duas tarefas adjacentes x e y que satisfazem as condições da Proposição 3.8.

3.3.2 Algoritmo EI

O Algoritmo 3.7 descreve o procedimento EI aplicado ao SMSPETP-SIS. Nele, a entrada UB corresponde a um limite superior conhecido para o problema. Esse limite é atualizado pelo procedimento EI de modo que o valor UB retornado é a solução ótima para o problema. No presente trabalho, o limite superior inicial é fornecido pelo algoritmo GVNS, apresentado na Seção 3.2.2.

Algoritmo 3.7 EI$(I, n, F(.), UB)$

```
seq(δ_0) ← ∅;
nseq(δ_0) ← I;
L ← {δ_0};
enquanto L ≠ ∅ faça
    δ ← o último nó inserido em L;
    L ← L \ {δ};
    se nseq(δ) = ∅ então
        se F(seq(δ)) < UB então
            UB ← F(seq(δ));
        fim
    senão
        para x ∈ nseq(δ) faça
            se a subsequência seq(δ) ∪ {x} não viola as condições de otimalidade então
                seq(δ_filho) ← seq(δ) ∪ {x};
                nseq(δ_filho) ← nseq(δ) \ {x};
                L ← L ∪ {δ_filho};
            fim
        fim
    fim
fim
Retorne UB;
```

Para exemplificar como o algoritmo EI funciona, considere um SMSPETP-SIS de 4 tarefas representado pela Tabela 3.3. Nessa tabela, a primeira coluna apresenta o conjunto de tarefas a serem programadas, isto é, $I = \{1, 2, 3, 4\}$. As demais colunas indicam o tempo de execução (P_x), o custo por unidade de antecipação (α_x), o custo por unidade de atraso (β_x), assim como as datas de início (E_x) e de término (T_x) da janela de conclusão de cada tarefa $x \in I$.

Um dos dados de entrada do algoritmo EI é um limite superior (UB) para o problema. Considere o valor de UB inicial dado pela sequência obtida ao se aplicar a heurística gulosa

Tabela 3.3: Problema-teste para exemplificar o algoritmo EI.

I	P_x	E_x	T_x	α_x	β_x
1	5	11	14	9	10
2	4	9	9	7	9
3	6	6	8	2	3
4	3	9	9	1	4

EDD (ver Seção 3.2.2.2), ou seja:

$$UB = F(3,2,4,1) = 0 + (1 \times \beta_2) + (4 \times \beta_4) + (4 \times \beta_1) = 0 + (1 \times 9) + (4 \times 4) + (4 \times 10) = 65$$

conforme os dados apresentados na Tabela 3.3. A Figura 3.11 apresenta a programação ótima da sequência obtida pela heurística EDD, bem como a soma das penalidades associadas a essa programação.

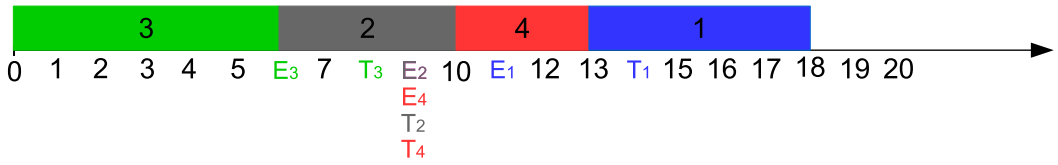
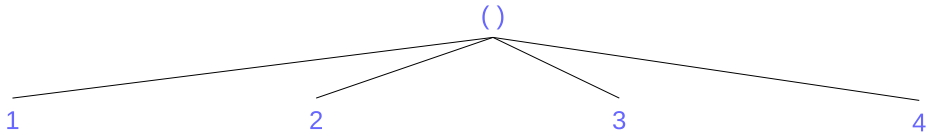


Figura 3.11: Solução obtida pela heurística gulosa EDD.

Para facilidade de entendimento, um nó δ será representado por $\frac{seq(\delta)}{nseq(\delta)}$. Seja L a lista de nós a serem investigados. O procedimento EI inicia com a inserção do nó $\frac{(\ )}{\{1,2,3,4\}}$ como o único elemento de L. Observa-se que esse nó armazena a subsequência vazia, representada por ( ), e o conjunto de tarefas ainda não sequenciadas, $\{1,2,3,4\}$.

O próximo passo do procedimento EI consiste em retirar e investigar o último nó de L, isto é, o nó $\frac{(\ )}{\{1,2,3,4\}}$. Como as subsequências de apenas uma tarefa não violam as condições de otimalidade, os nós que armazenam essas subsequências são inseridos em L. A Figura 3.12 ilustra o segundo passo do procedimento EI.



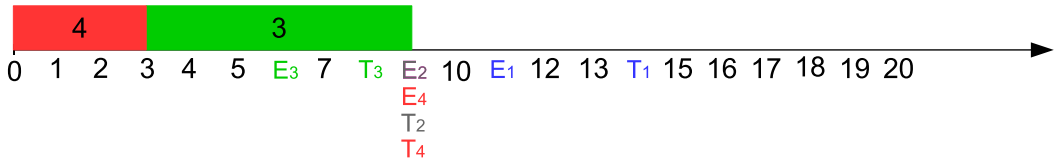
- $UB = 65$
- $L = \left\{\frac{(1)}{\{2,3,4\}}, \frac{(2)}{\{1,3,4\}}, \frac{(3)}{\{1,2,4\}}, \frac{(4)}{\{1,2,3\}}\right\}$

Figura 3.12: Segundo passo do procedimento EI.

O último nó corrente de L, nó $\frac{(4)}{\{1,2,3\}}$, é, então, retirado e investigado. As subsequências $(4,1)$ e $(4,2)$ não violam as condições de otimalidade e, portanto, os nós que armazenam essas

subsequências são inseridos em L. Por outro lado, a subsequência $(4, 3)$ viola a condição de otimalidade (iii) do Corolário 3.2. De fato, a primeira tarefa da programação ótima dessa subsequência, tarefa 4, inicia sua execução na data 0 e as tarefas 4 e 3 são tais que $soma_1 = 9 > 0 = soma_2$. Logo, o nó que armazena a subsequência $(4, 3)$ não é inserido em L. A Figura 3.13 ilustra a análise da subsequência $(4, 3)$, ao passo que a Figura 3.14 apresenta a árvore de busca após a investigação do nó $\frac{(4)}{\{1,2,3\}}$.

$$soma_1 = (6 \times \alpha_4) + (1 \times \beta_3) = (6 \times 1) + (1 \times 3) = 9$$



$$soma_2 = 0 + 0 = 0$$

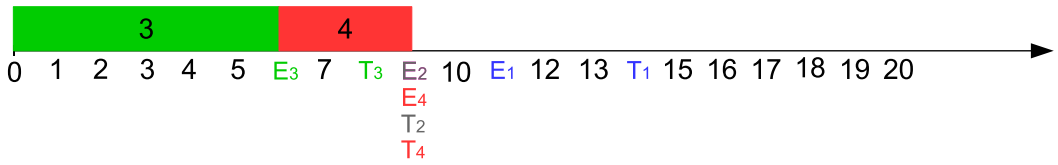
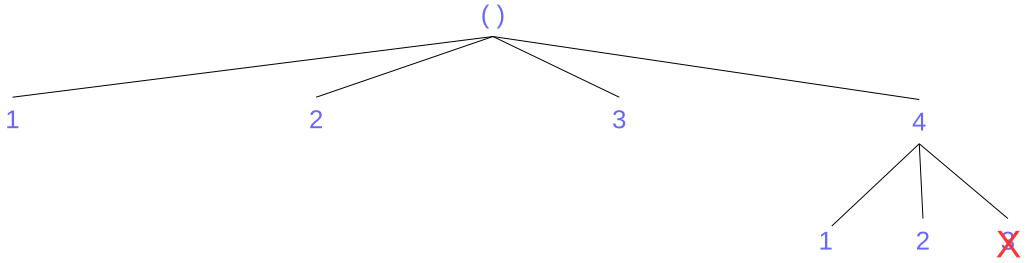


Figura 3.13: Análise da subsequência $(4, 3)$.

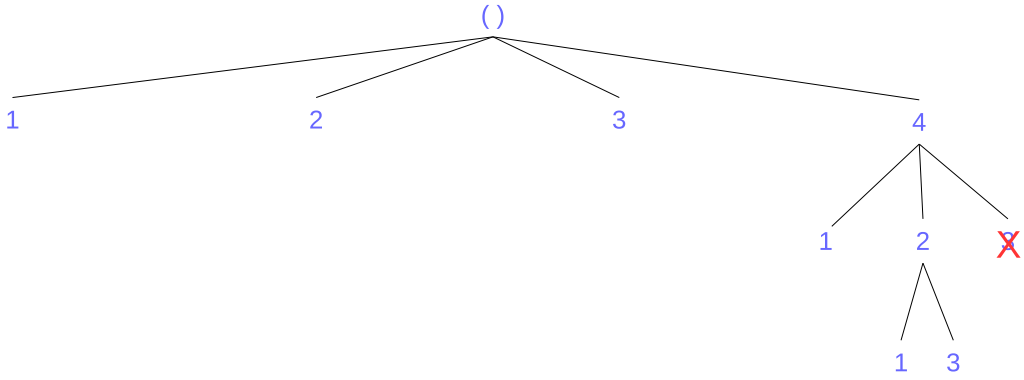


- $UB = 65$
- $L = \left\{ \frac{(1)}{\{2,3,4\}}, \frac{(2)}{\{1,3,4\}}, \frac{(3)}{\{1,2,4\}}, \frac{(4,1)}{\{2,3\}}, \frac{(4,2)}{\{1,3\}} \right\}$

Figura 3.14: Árvore de busca após a investigação do nó $\frac{(4)}{\{1,2,3\}}$.

O próximo passo é retirar de L e investigar o nó $\frac{(4,2)}{\{1,3\}}$. As subsequências $(4, 2, 1)$ e $(4, 2, 3)$ não violam as condições de otimalidade. Logo, os nós que armazenam essas subsequências são inseridos em L (Figura 3.15).

A seguir, o nó $\frac{(4,2,3)}{\{1\}}$ é retirado de L e investigado. A subsequência $(4, 2, 3, 1)$ corresponde a uma sequência completa. Essa sequência é, então, avaliada. Como $F(4, 2, 3, 1) = 75 \geq 65 = UB$, o valor de UB não é atualizado. A Figura 3.16 apresenta a programação ótima da sequência $(4, 2, 3, 1)$, ao passo que a Figura 3.17 apresenta a árvore de busca após a investigação do nó $\frac{(4,2,3)}{\{1\}}$.

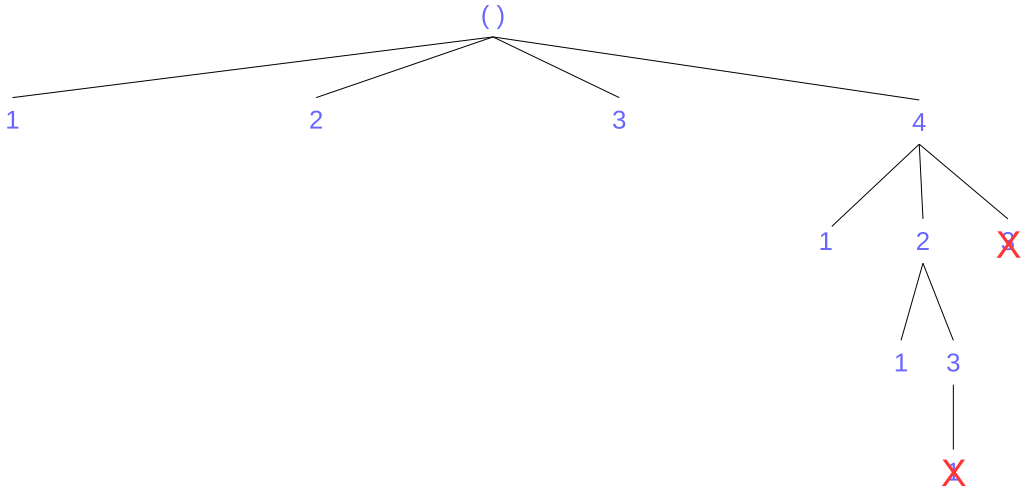


- $UB = 65$
- $L = \left\{ \frac{(1)}{\{2,3,4\}}, \frac{(2)}{\{1,3,4\}}, \frac{(3)}{\{1,2,4\}}, \frac{(4,1)}{\{2,3\}}, \frac{(4,2,1)}{\{3\}}, \frac{(4,2,3)}{\{1\}} \right\}$

Figura 3.15: Árvore de busca após a investigação do nó $\frac{(4,2)}{\{1,3\}}$.

$$F(4,2,3,1) = (6 \times \alpha_4) + (2 \times \alpha_2) + (5 \times \beta_3) + (4 \times \beta_1) = (6 \times 1) + (2 \times 7) + (5 \times 3) + (4 \times 10) = 75$$

Figura 3.16: Programação ótima da sequência $(4, 2, 3, 1)$.



- $UB = 65$
- $L = \left\{ \frac{(1)}{\{2,3,4\}}, \frac{(2)}{\{1,3,4\}}, \frac{(3)}{\{1,2,4\}}, \frac{(4,1)}{\{2,3\}}, \frac{(4,2,1)}{\{3\}} \right\}$

Figura 3.17: Árvore de busca após a investigação do nó $\frac{(4,2,3)}{\{1\}}$.

O próximo passo é retirar de L e investigar o nó $\frac{(4,2,1)}{\{3\}}$. A subsequência $(4, 2, 1, 3)$ corresponde a uma sequência completa de I. Essa sequência é, então, avaliada. Como $F(4, 2, 1, 3) = 40 < 65 = UB$, o valor de UB é atualizado para $UB = F(4, 2, 1, 3) = 40$. A Figura 3.18 apresenta a

programação ótima da sequência $(4, 2, 1, 3)$, ao passo que a Figura 3.19 apresenta a árvore de busca após a investigação do nó $\frac{(4,2,1)}{\{3\}}$.

$$F(4,2,1,3) = (4 \times \alpha_4) + 0 + 0 + (12 \times \beta_3) = (4 \times 1) + 0 + 0 + (12 \times 3) = 40$$

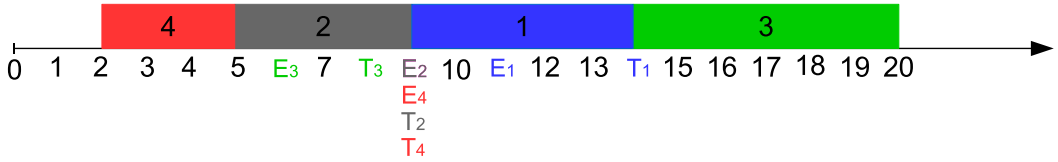
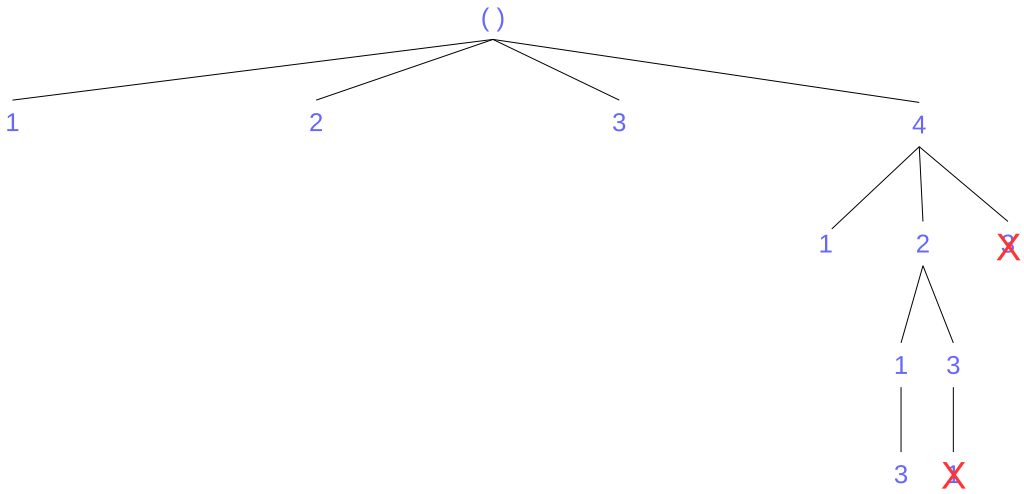


Figura 3.18: Programação ótima da sequência $(4, 2, 1, 3)$.



- $UB = 40$
- $L = \left\{ \frac{(1)}{\{2,3,4\}}, \frac{(2)}{\{1,3,4\}}, \frac{(3)}{\{1,2,4\}}, \frac{(4,1)}{\{2,3\}} \right\}$

Figura 3.19: Árvore de busca após a investigação do nó $\frac{(4,2,1)}{\{3\}}$.

Em seguida, o nó $\frac{(4,1)}{\{2,3\}}$ é retirado de L e investigado. A subsequência $(4, 1, 2)$ viola a condição de otimalidade (i), pois $F(4, 1, 2) = 57 \geq 40 = UB$ (Figura 3.20). Logo, o nó que armazena essa subsequência não é inserido em L. Por outro lado, a subsequência $(4, 1, 3)$ não viola as condições de otimalidade e, portanto, o nó que armazena a subsequência $(4, 1, 3)$ é inserido em L. A Figura 3.21 mostra a árvore de busca após o nó $\frac{(4,1)}{\{2,3\}}$ ser investigado.

$$F(4,1,2) = (3 \times \alpha_4) + 0 + (6 \times \beta_2) = (3 \times 1) + 0 + (6 \times 9) = 57$$

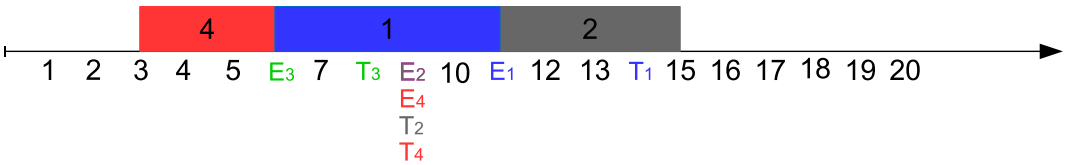
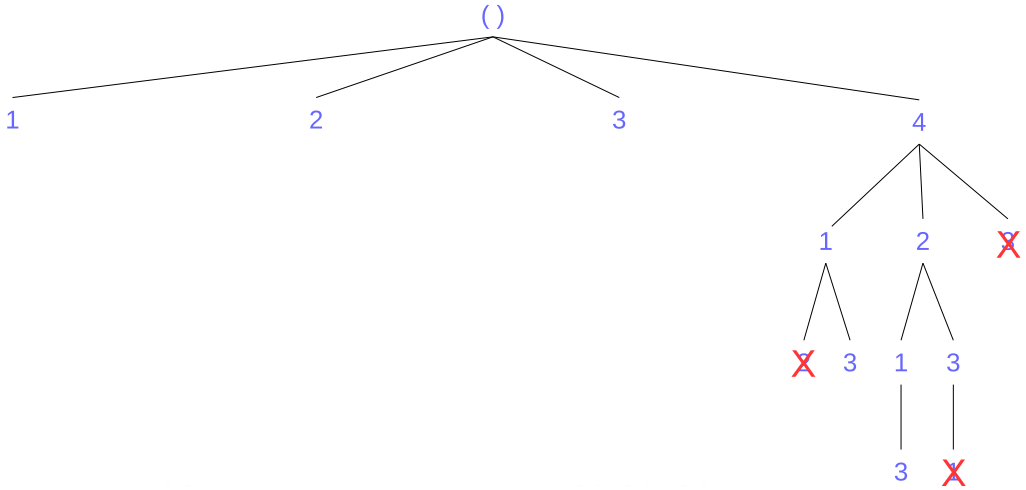


Figura 3.20: Análise da subsequência $(4, 1, 2)$.

A investigação dos nós de L continua até chegar o momento em que o nó $\frac{(2,4)}{\{1,3\}}$ é investigado. A subsequência $A = (2, 4, 1)$ viola a condição de otimalidade (ii). De fato, as tarefas 2 e 4 na programação ótima de A, π^*_A, são tais que $C_4^{\pi^*_A} = 9 \leq 9 = \min(9, 9) = \min(E_2, E_4)$ e

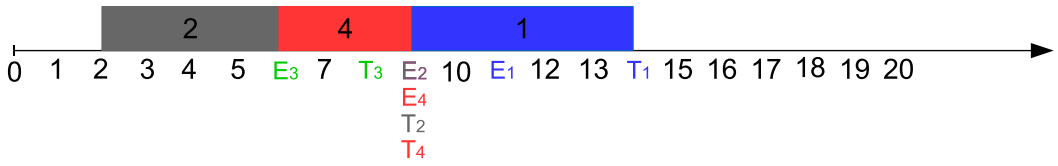


- $UB = 40$
- $L = \left\{ \frac{(1)}{\{2,3,4\}}, \frac{(2)}{\{1,3,4\}}, \frac{(3)}{\{1,2,4\}}, \frac{(4,1,3)}{\{2\}} \right\}$

Figura 3.21: Árvore de busca após a investigação do nó $\frac{(4,1)}{\{2,3\}}$.

$P_2 \times \alpha_4 = 4 \times 1 < 3 \times 7 = P_4 \times \alpha_2$. Como a troca da ordem de execução das tarefas 2 e 4 na subsequência $(2, 4, 1)$ reduz a soma das penalidades associadas a essa subsequência (Figura 3.22), bem como das sequências construídas a partir dela, o nó que armazena a subsequência $(2, 4, 1)$ não é inserido em L. Por outro lado, a subsequência $(2, 4, 3)$ viola a condição de otimalidade (i), pois $F(2, 4, 3) = 42 > 40 = UB$ (Figura 3.23). Portanto, o nó que armazena a subsequência $(2, 4, 3)$ também não é inserido em L. A Figura 3.24 apresenta a árvore de busca após a investigação do nó $\frac{(2,4)}{\{1,3\}}$.

$$soma_1 = (3 \times \alpha_2) + 0 + 0 = 3 \times 7 = 21$$



$$soma_2 = (4 \times \alpha_4) + 0 = 4 \times 1 = 4$$

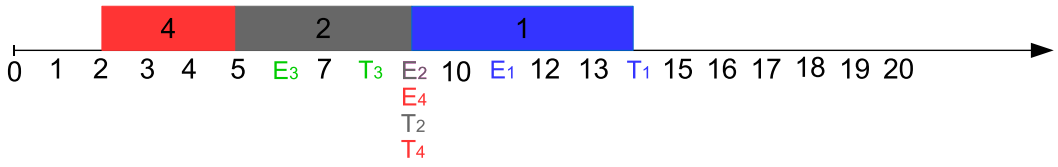


Figura 3.22: Análise da subsequência $(4, 3)$.

O procedimento EI é interrompido quando não houver mais nós para serem investigados, ou seja, quando $L = \{\ \}$. O valor final de UB ($UB = 40$) é a solução ótima para o problema, ao passo que a sequência associada a esse valor, sequência $(4, 2, 1, 3)$, é uma sequência ótima. A Figura 3.25 apresenta a árvore de busca após todos os nós terem sido investigados.

$$F(2,4,3) = (3 \times \alpha_2) + 0 + (7 \times \beta_3) = (3 \times 7) + 0 + (7 \times 3) = 42$$

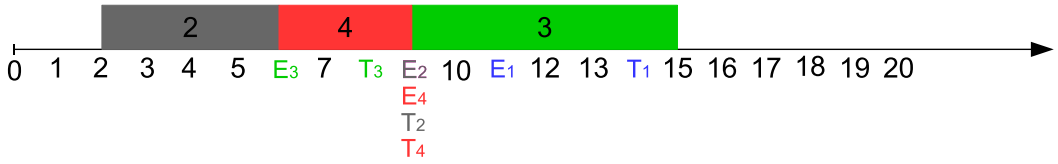
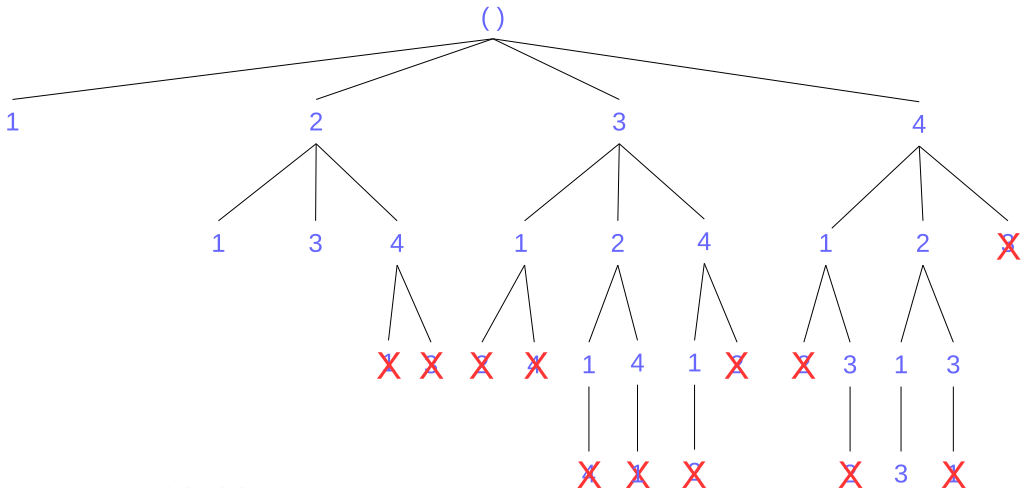
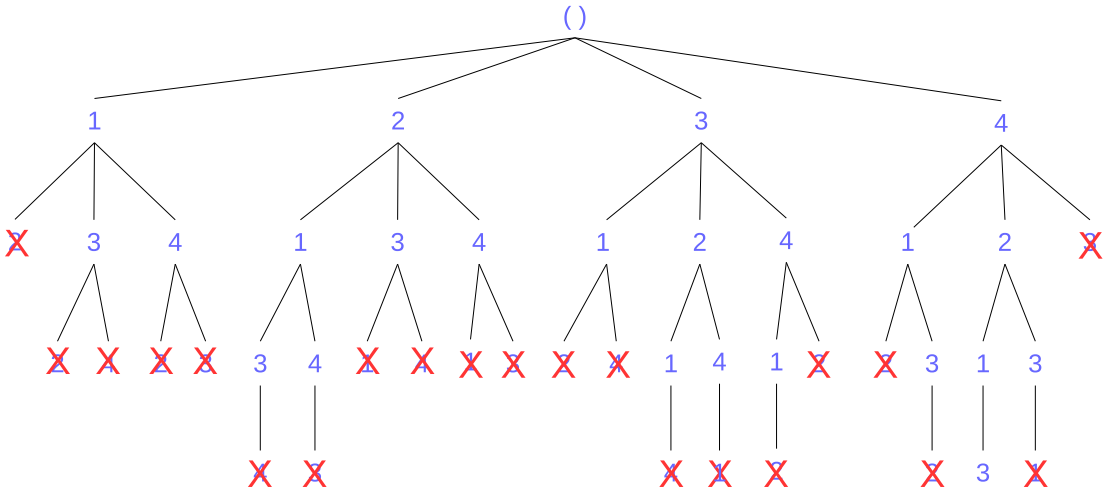


Figura 3.23: Análise da subsequência $(2,4,3)$.



- $UB = 40$
- $L = \left\{ \frac{(1)}{\{2,3,4\}}, \frac{(2,1)}{\{3,4\}}, \frac{(2,3)}{\{1,4\}} \right\}$

Figura 3.24: Árvore de busca após a investigação do nó $\frac{(2,4)}{\{1,3\}}$.



- $UB = 40$
- $L = \{\ \}$

Figura 3.25: Árvore de busca após todos os nós terem sido investigados.

3.4 Resultados computacionais

Nesta seção são apresentados os resultados computacionais obtidos com os algoritmos descritos nas Seções 3.1, 3.2 e 3.3.

O conjunto de problemas-teste descrito na Seção 2.2 foi utilizado na realização dos experi-

mentos computacionais. Os algoritmos OTA, AAOTO, OTA', GVNS_r, GVNS_f, GVNS_{rf} e EI apresentados nas Seções 3.1.1 3.1.2, 3.1.3, 3.2.2 e 3.3, respectivamente, foram implementados em C++, sendo utilizado o compilador g++, versão 4.8.5, para a sua execução.

Os experimentos foram realizados em um computador Intel® Xeon(R) CPU E5620 @ 2.40GHz × 16, com 48 GB de RAM e sistema operacional CentOS Linux 7. Apesar de o processador do equipamento utilizado possuir mais de um núcleo, os algoritmos não foram otimizados para multiprocessamento.

Para auxiliar na apresentação e comparação dos resultados, utilizou-se a métrica desvio relativo médio, dada por $\text{DRM}_i^{\text{avg}} = \left(\overline{f}_i^A - f_i^\star\right)/f_i^\star$. Dado um problema-teste i, f_i^* representa o valor da melhor solução conhecida para esse problema, enquanto $\overline{f}_i^A$ representa a média dos valores das soluções encontradas pelo algoritmo dado A. Logo, a métrica $\text{DRM}_i^{\text{avg}}$ compara a média dos resultados obtidos pelo algoritmo A com a melhor solução conhecida para o respectivo problema (quanto menor for o valor de $\text{DRM}^{\text{avg}_i}$, melhor será a solução média encontrada).

Neste trabalho, foram consideradas as soluções encontradas nos experimentos realizados com a melhor versão do Algoritmo Genético Adaptativo de Ribeiro *et al.* (2010) e com o melhor algoritmo de Rosa *et al.* (2017) para determinar o valor de referência f_i^* de um dado problema-teste i. Esses algoritmos foram implementados nas mesmas condições que os algoritmos propostos neste trabalho.

O modelo de programação matemática de Gomes Júnior *et al.* (2007) (vide Seção 4.2.1) também foi implementado a fim de validar os métodos propostos, bem como ser utilizado como referência para comparações. Foram utilizados a *C++ Concert Technology* e o otimizador *IBM ILOG CPLEX Optimization Studio 12.6.2* para a implementação e resolução desse modelo. O CPLEX foi configurado para utilizar somente uma *thread* e o tempo de execução foi limitado em uma hora para resolver cada problema-teste. Os demais parâmetros do CPLEX não foram alterados. Para cada problema-teste resolvido com os algoritmos GVNS_r, GVNS_f e GVNS_{rf}, a melhor solução encontrada foi fornecida ao CPLEX.

O restante desta seção está organizado da seguinte maneira. Os algoritmos OTA, OTA' e AAOTO são comparados na Seção 3.4.1. Na Seção 3.4.2 é realizado o ajuste do parâmetro VNDmax, presente nos algoritmos GVNS_r e GVNS_{rf}, e do parâmetro VNSmax, presente nos algoritmos GVNS_r, GVNS_f e GVNS_{rf}. Os resultados obtidos com os algoritmos GVNS_r, GVNS_f, GVNS_{rf} e EI aplicados a problemas-teste do SMSPETP-SIS com até 20 tarefas são analisados na Seção 3.4.3. Enfim, os resultados obtidos com os algoritmos GVNS_r, GVNS_f e GVNS_{rf} aplicados a problemas-teste do SMSPETP-SDS com até 75 tarefas são discutidos na Seção 3.4.4.

3.4.1 Comparação OTA × AAOTO × OTA'

O algoritmo GVNS_f (vide Seção 3.2.2) foi aplicado aos problemas-teste de 10, 20, 30, 40, 50 e 75 tarefas a fim de comparar a eficiência dos algoritmos de programação ótima de uma dada sequência de execução de tarefas apresentados na Seção 3.1. O parâmetro GVNSmax do GVNS_f foi fixado em $2n$, em que n representa o número de tarefas a serem programadas. Devido à natureza estocástica do algoritmo GVNS_f, a semente de números aleatórios foi fixada para cada problema-teste. Desse modo, para cada problema-teste, a sequência de tarefas geradas pelo GVNS_f é sempre a mesma, independente do algoritmo utilizado para avaliar tais sequências. Cada problema-teste foi resolvido três vezes, sendo utilizado um dos algoritmos OTA, AAOTO e OTA' para avaliar as sequências de execução geradas pelo GVNS_f em cada uma dessas vezes.

A Tabela 3.4 apresenta os tempos médios demandados pelo algoritmo GVNS_f associado aos respectivos algoritmos de programação ótima de uma dada sequência de execução de tarefas apresentados na Seção 3.1. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. As demais colunas apresentam os tempos médios exigidos pelo

$GVNS_f$ ao se utilizar os algoritmos OTA, AAOTO e OTA', respectivamente, para avaliar cada sequência de tarefas gerada durante a resolução dos respectivos problemas-teste.

Tabela 3.4: Tempos médios (em segundos) demandados pelo algoritmo $GVNS_f$ associado aos algoritmos OTA, AAOTO e OTA', respectivamente.

n	OTA	AAOTO	OTA'
10	0,04	0,02	0,03
20	1,43	0,72	0,92
30	16,26	6,91	9,83
40	103,52	40,92	56,37
50	405,07	149,01	199,18
75	6279,69	1933,78	2676,16

De acordo com a Tabela 3.4, os tempo médios demandados pelo algoritmo $GVNS_f$ com o OTA ($GVNS_f$/OTA) foram significativamente superiores que os respectivos tempos médios demandados pelo $GVNS_f$ com o AAOTO ($GVNS_f$/AAOTO) e com o OTA' ($GVNS_f$/OTA'). Para os problemas-teste com 75 tarefas, por exemplo, o tempo médio demandado pelo $GVNS_f$/OTA foi maior que o dobro do tempo médio demandado pelo $GVNS_f$/OTA' e maior que o triplo do tempo médio demandado pelo $GVNS_f$/AAOTO. Ainda de acordo com a Tabela 3.4, os tempos médios demandados pelo $GVNS_f$/OTA' também foram maiores que os respectivos tempos médios exigidos pelo $GVNS_f$/AAOTO. Por exemplo, para os problemas-teste com 75 tarefas, o tempo médios demandado pelo $GVNS_f$/OTA' ultrapassa em mais de 38% o tempo médio demandado pelo $GVNS_f$/AAOTO.

Visto que o AAOTO foi o algoritmo de programação ótima de uma dada sequência execução de tarefas que fez com que o $GVNS_f$ demandasse menores tempos de execução, ele foi utilizado para avaliar cada sequência de tarefas gerada pelos algoritmos $GVNS_r$, $GVNS_f$, $GVNS_{rf}$ e EI nos demais experimentos computacionais.

3.4.2 Ajuste dos parâmetros dos algoritmos propostos

O conjunto de problemas-teste de 50 tarefas foi utilizado para calibrar o parâmetro VNDmax da busca local VND_r. O conjunto de problemas-teste com 50 tarefas foi escolhido por ser um conjunto de problemas de tamanho intermediário. Foram experimentados os valores $5n$, $7n$, $9n$, $11n$ e $13n$ para VNDmax, em que n representa o número de tarefas a serem programadas. Cada um dos 16 problemas-teste foi resolvido 30 vezes, sendo o VND_r aplicado à solução inicial proposta. Os resultados obtidos são apresentados na Tabela 3.5. Nessa tabela, para cada coluna associada a um valor atribuído à VNDmax, a coluna $\overline{\text{DRM}^{\text{avg}}}$ apresenta a média dos $\text{DRM}_i^{\text{avg}}$ (em porcentagem) obtidos, enquanto a coluna $\overline{\text{t}}$ apresenta o tempo médio (em segundos) demandado para resolver os problemas-teste.

Tabela 3.5: Influência do parâmetro VNDmax no procedimento VND_r aplicado aos problemas-teste com 50 tarefas.

3n		**5n**		**7n**		**9n**		**11n**		**13n**	
$\overline{\text{DRM}^{\text{avg}}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}^{\text{avg}}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}^{\text{avg}}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}^{\text{avg}}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}^{\text{avg}}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}^{\text{avg}}}$ (%)	$\overline{\text{t}}$ (s)
15,19	0,03	11,63	0,04	9,63	0,05	8,33	0,06	7,63	0,08	7,19	0,09

De acordo com a Tabela 3.5, como era esperado, à medida em que se aumenta o valor de VNDmax, menor é média dos desvios relativos médios das soluções obtidas e maior é o tempo

médio demandado pelo VND_r. A diferença entre o valor de $\overline{\text{DRM}}^{\text{avg}}$ com VNDmax $= 11n$ e o valor de $\overline{\text{DRM}}^{\text{avg}}$ com VNDmax $= 13n$ é menor que 0,5 pontos percentuais. Além disso, o tempo médio demandado com VNDmax $= 13n$ é 0,01 segundos superior ao tempo médio demandado com VNDmax $= 11n$. Dado que a busca local VND_r é aplicada um grande número de vezes dentro do algoritmo GVNS_r, o valor de VNDmax foi fixado em $11n$ nos demais experimentos.

O conjunto de problemas-teste com 30 tarefas foi utilizado para calibrar o VNSmax dos algoritmos GVNS_r, GVNS_f e GVNS_{rf}. Optou-se por utilizar o conjunto de problemas-teste com 30 tarefas por ser um conjunto de problemas de tamanho intermediário e com problemas-testes diferentes dos utilizados para calibrar o parâmetro VNDmax. Foram experimentados os valores n, $2n$, $3n$, $4n$ e $5n$ para VNSmax, em que n representa o número de tarefas a serem programadas. Novamente, cada um dos 16 problemas-teste foi resolvido 30 vezes pelo GVNS_f e os resultados obtidos são resumidos na Tabela 3.6. Nessa tabela, para cada coluna associada a um valor atribuído à VNSmax, a coluna $\overline{\text{DRM}}^{\text{avg}}$ apresenta a média dos $\text{DRM}_i^{\text{avg}}$ (em porcentagem) obtidos, ao passo que a coluna $\overline{\text{t}}$ apresenta o tempo médio (em segundos) demandado para resolver os problemas-teste.

Tabela 3.6: Influência do parâmetro VNSmax no procedimento GVNS_f aplicado aos problemas-teste com 30 tarefas.

$1n$		$2n$		$3n$		$4n$		$5n$	
$\overline{\text{DRM}}^{\text{avg}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}}^{\text{avg}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}}^{\text{avg}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}}^{\text{avg}}$ (%)	$\overline{\text{t}}$ (s)	$\overline{\text{DRM}}^{\text{avg}}$ (%)	$\overline{\text{t}}$ (s)
0,13	3,91	0,08	7,09	0,08	10,07	0,07	13,35	0,07	16,24

Segundo a Tabela 3.6, como também era esperado, à medida em que se aumenta o valor de VNSmax, menor é média dos desvios relativos médios das soluções obtidas e maior é o tempo médio demandado. Para os valores de VNSmax maiores que ou iguais a $2n$, as diferenças entre os respectivos $\overline{\text{DRM}}^{\text{avg}}$ obtidos são menores que ou iguais a 0,02 pontos percentuais. Logo, considerando-se, também, que o tempo médio observado aumenta consideravelmente com o aumento do valor de VNSmax, o valor de VNSmax foi fixado em $2n$ nos demais experimentos realizados com os algoritmos GVNS_r, GVNS_f e GVNS_{rf}.

3.4.3 Resultados com os problemas-teste do SMSPETP-SIS com até 20 tarefas

Nesta seção são discutidos os resultados computacionais referentes à aplicação dos algoritmos GVNS_r, GVNS_f, GVNS_{rf} e EI aos problemas-teste do SMSPETP-SIS com até 20 tarefas. Logo, as matrizes de tempos de preparação da máquina presentes nos problemas-teste foram desconsideradas.

Devido à natureza estocástica dos algoritmos GVNS_r, GVNS_f e GVNS_{rf}, bem como dos algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017), cada problema-teste foi resolvido 30 vezes por cada um desses algoritmos. Para cada problema-teste, a melhor solução encontrada por esses algoritmos foi fornecida ao algoritmo EI.

A Tabela 3.7 mostra os tempos médios demandados (em segundos) pelos algoritmos GVNS_r, GVNS_f e GVNS_{rf}, bem como pelos melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017) e pelo CPLEX, em cada conjunto de problemas-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. Exceto para o CPLEX, que foi utilizado para resolver cada problema-teste uma única vez, as médias apresentadas consideram as 30 aplicações dos respectivos algoritmos em cada problema-teste.

De acordo com a Tabela 3.7, o CPLEX foi capaz de resolver apenas os problemas-teste com até 10 tarefas dentro do limite de tempo de uma hora de execução. O algoritmo de Ribeiro

Tabela 3.7: Tempos médios (em segundos) demandados pelos melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017), pelos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$, bem como pelo CPLEX, para resolver os problemas-teste com até 20 tarefas do SMSPETP-SIS.

n	Ribeiro 2010	Rosa 2017	GVNS$_r$	GVNS$_f$	GVNS$_{rf}$	CPLEX[1]
06	0,18	0,01	0,00	0,00	0,00	0,06
07	0,21	0,01	0,01	0,00	0,01	0,54
08	0,26	0,01	0,01	0,01	0,01	3,11
09	0,32	0,02	0,02	0,01	0,02	23,45
10	0,39	0,03	0,02	0,02	0,02	250,54
11	0,49	0,04	0,03	0,03	0,03	–
12	0,63	0,06	0,04	0,05	0,04	–
13	0,75	0,08	0,06	0,07	0,06	–
14	0,93	0,10	0,08	0,11	0,08	–
15	1,16	0,14	0,12	0,17	0,11	–
16	1,40	0,17	0,15	0,23	0,13	–
17	1,86	0,22	0,19	0,34	0,17	–
18	2,01	0,29	0,24	0,49	0,22	–
19	2,44	0,34	0,27	0,58	0,25	–
20	3,20	0,47	0,36	0,84	0,33	–

[1]O tempo de execução do CPLEX se mostrou proibitivo para resolver os problemas-teste com mais de 10 tarefas. Por esse motivo, esta tabela não contém as informações desses problemas.

et al. (2010) demandou um tempo médio superior aos respectivos tempos médios demandados pelo algoritmo de Rosa *et al.* (2017) e pelos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$. Os tempos médios demandados pelo algoritmo de Rosa *et al.* (2017) e pelos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ são todos inferiores a um segundo, ao passo que o algoritmo de Ribeiro *et al.* (2010) levou um tempo médio de 3,2 segundos para resolver os problemas-teste com 20 tarefas.

Embora não esteja apresentado em nenhuma tabela, os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$, bem como os algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017), foram capazes de encontrar as respectivas soluções ótimas de todos os problemas em que tais soluções são conhecidas, isto é, de todos os problemas-teste que foram resolvidos pelo CPLEX ou pelo algoritmo EI. Além disso, somente o algoritmo GVNS$_f$ não obteve soluções de desvios relativos médios iguais a zero para todos os problemas-teste. Em outras palavras, para cada problema-teste, os algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017) e os algoritmos GVNS$_r$ e GVNS$_{rf}$ encontraram sempre as mesmas soluções, sendo tais soluções iguais para todos esses algoritmos. No entanto, os respectivos desvios relativos médios das soluções encontradas pelo algoritmo GVNS$_f$ são menores que ou iguais a 0,05%.

O tempo de execução do otimizador CPLEX se mostrou proibitivo para resolver os problemas-teste com mais de 10 tarefas. Comportamento similar foi observado com o algoritmo EI em problemas-teste com mais de 15 tarefas. A Tabela 3.8 mostra os tempos médios demandados (em segundos) pelo algoritmo EI e pelo CPLEX para cada conjunto de problemas-teste solucionado. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste.

A Tabela 3.8 mostra que os tempos médios demandados pelo CPLEX foram muito superiores àqueles requeridos pelo algoritmo EI. Para o conjunto de problemas-teste com 10 tarefas, por exemplo, enquanto o EI demandou tempo médio igual a 0,07 segundos, o CPLEX demandou 250 segundos. Somente o algoritmo EI foi capaz de resolver os problemas-teste com até 15 tarefas dentro do limite de tempo de uma hora de execução. O CPLEX possibilitou resolver

Tabela 3.8: Tempos médios (em segundos) demandados pelo algoritmo EI e pelo CPLEX para resolver os problemas-teste com até 15 tarefas do SMSPETP-SIS.

n	EI	CPLEX[1]
06	0,00	0,06
07	0,00	0,54
08	0,01	3,11
09	0,02	23,45
10	0,07	250,54
11	0,39	–
12	2,52	–
13	28,54	–
14	116,80	–
15	799,76	–

[1]O tempo de execução do CPLEX se mostrou proibitivo para resolver os problemas-teste com mais de 10 tarefas. Por esse motivo, esta tabela não contém as informações desses problemas.

apenas os problemas-teste com até 10 tarefas dentro desse limite de tempo.

3.4.4 Resultados com os problemas-teste do SMSPETP-SDS com até 75 tarefas

Os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ foram comparados com a melhor versão do Algoritmo Genético Adaptativo de Ribeiro *et al.* (2010) e com o melhor algoritmo de Rosa *et al.* (2017). Foram utilizados os problemas-teste do SMSPETP-SDS envolvendo 10, 20, 30, 40 50 e 75 tarefas. Dado que cada conjunto de problemas-teste com o mesmo número de tarefas contém 16 problemas, foram utilizados um total de 96 problemas-teste.

Cada problema-teste foi resolvido 30 vezes com cada algoritmo implementado, exceto aqueles com 75 tarefas, que foram resolvidos apenas 10 vezes em virtude do maior tempo computacional demandado. Os resultados obtidos são apresentados e discutidos a seguir.

Primeiro, com base na Tabela 3.9, discute-se a qualidade das soluções médias encontradas por cada algoritmo. Os testes estatísticos apresentados no Apêndice foram utilizados para comparar as soluções obtidas pelos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ com as respectivas soluções obtidas pelos melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017). Os resultados obtidos por esses testes estatísticos são apresentados nas Tabelas 3.10–3.13. Finalmente, com base na Tabela 3.14, analisam-se os tempos médios demandados por cada algoritmo em cada conjunto de problemas-teste com o mesmo número de tarefas.

3.4.4.1 Desvios médios relativos das soluções soluções encontradas

A Tabela 3.9 apresenta as médias dos desvios relativos médios (em porcentagem) das soluções obtidas com os melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017) e com os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ para os respectivos problemas-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. A fim de validar os algoritmos heurísticos implementados, a Tabela 3.9 também apresenta a média dos desvios médios relativos das soluções encontradas pelo CPLEX para os respectivos problemas-teste com 10 tarefas. O tempo de execução do CPLEX mostrou-se proibitivo para resolver os problemas-teste com mais de 10 tarefas. Por esse motivo, as informações associadas a esses problemas não são apresentadas.

Tabela 3.9: Médias dos desvios relativos médios (em porcentagem) das soluções obtidas com os melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017), com os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ e com o CPLEX.

n	Ribeiro (2010)	Rosa (2017)	GVNS$_r$	GVNS$_f$	GVNS$_{rf}$	CPLEX
10	0,00	0,00	0,00	0,00	0,00	0,00
20	0,10	0,01	0,04	0,13	0,04	–
30	0,30	0,20	0,17	0,08	0,17	–
40	0,46	0,25	0,15	0,03	0,15	–
50	0,77	0,53	0,28	0,05	0,28	–
75	1,85	0,99	0,56	-0,17	0,54	–

De acordo com a Tabela 3.9, todos os algoritmos implementados encontraram a solução ótima em todas as execuções para os problemas-teste com 10 tarefas, pois todas as médias dos desvios relativos médios das soluções encontradas para os respectivos problemas desse conjunto são iguais a zero por cento, inclusive das soluções encontradas pelo CPLEX. Ainda de acordo com a Tabela 3.9, o algoritmo GVNS$_f$ foi o que obteve melhor desempenho com relação à qualidade das soluções. Exceto para os problemas-teste com 20 tarefas, ele superou todos os outros algoritmos, incluindo os melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017). Observa-se, também, que o algoritmo GVNS$_f$ foi capaz de encontrar soluções médias melhores que as respectivas melhores soluções de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017) para problemas-teste com 75 tarefas. Os algoritmos GVNS$_r$ e GVNS$_{rf}$ demonstraram desempenhos bem similares. De fato, ambos obtiveram soluções médias melhores que as respectivas soluções médias dos melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017), exceto para os problemas-teste com 20 tarefas. O algoritmo de Ribeiro *et al.* (2010) foi o que apresentou pior desempenho entre os algoritmos experimentados.

3.4.4.2 Testes estatísticos com o melhor algoritmo de Ribeiro *et al.* (2010)

Para verificar o número de vezes em que a melhor versão do Algoritmo Genético Adaptativo de Ribeiro *et al.* (2010) obteve melhor desempenho que os algoritmos propostos neste trabalho, os testes de hipóteses paramétrico e de aleatoriedade foram realizados com os algoritmos dispostos na forma Ribeiro *et al.* (2010) $\times$ GVNS$_r$, Ribeiro *et al.* (2010) $\times$ GVNS$_f$ e Ribeiro *et al.* (2010) $\times$ GVNS$_{rf}$, respectivamente. Com essa disposição, rejeitar a hipótese nula permite afirmar que, com um nível de significância igual a 0,05, há evidências estatísticas de que a solução média obtida pelo melhor algoritmo de Ribeiro *et al.* (2010) é melhor que a solução média obtida com o respetivo algoritmo na comparação.

As colunas "Teste Paramétrico" e "Teste de Aleatoriedade" da Tabela 3.10 mostram os resultados obtidos ao se aplicar os testes de hipóteses nas soluções de cada problema-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. As colunas "GVNS$_r$", "GVNS$_f$" e "GVNS$_{rf}$" apresentam o número de vezes em que a hipótese nula foi rejeitada ao se aplicar os correspondes testes de hipóteses com as disposições Ribeiro *et al.* (2010) $\times$ GVNS$_r$, Ribeiro *et al.* (2010) $\times$ GVNS$_f$ e Ribeiro *et al.* (2010) $\times$ GVNS$_{rf}$, respectivamente.

De acordo com a Tabela 3.10, foram obtidos os mesmos resultados para os testes de hipóteses utilizados (teste paramétrico e teste de aleatoriedade). Nenhum deles rejeitou a hipótese nula nas comparações associadas a problemas-teste com até 10, 40, 50 e 75 tarefas. Ambos os testes concluíram que a solução média obtida pelo melhor algoritmo de Ribeiro *et al.* (2010) é melhor que as respectivas soluções médias obtidas pelos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ em apenas um problema-teste com 30 tarefas. Já no conjunto de problemas-teste com 20 tarefas, a

Tabela 3.10: Número de vezes em que o melhor algoritmo de Ribeiro *et al.* (2010) é melhor que os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$.

n	Teste paramétrico			Teste de aleatoriedade		
	GVNS$_r$	**GVNS$_f$**	**GVNS$_{rf}$**	**GVNS$_r$**	**GVNS$_f$**	**GVNS$_{rf}$**
10	0	0	0	0	0	0
20	0	1	0	0	1	0
30	1	1	1	1	1	1
40	0	0	0	0	0	0
50	0	0	0	0	0	0
75	0	0	0	0	0	0
Total	1	2	1	1	2	1

hipótese nula foi rejeitada uma vez na comparação Ribeiro *et al.* (2010) × GVNS$_f$ e nenhuma vez nas demais comparações.

Os testes de hipóteses paramétrico e de aleatoriedade também foram utilizados para verificar o número de vezes em que os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ obtiveram soluções médias melhores que as respectivas soluções médias obtidas com o melhor algoritmo de Ribeiro *et al.* (2010). Para isso, esses testes foram realizados também com as disposições GVNS$_r$×Ribeiro *et al.* (2010), GVNS$_f$×Ribeiro *et al.* (2010) e GVNS$_{rf}$×Ribeiro *et al.* (2010). Com essas disposições, rejeitar a hipótese nula significa que, com um nível de significância igual a 0,05, há evidências estatísticas suficientes para afirmar que a solução média obtida pelo melhor algoritmo de Ribeiro *et al.* (2010) é pior que a solução média obtida com o respetivo algoritmo na comparação.

As colunas "Teste Paramétrico" e "Teste de Aleatoriedade" da Tabela 3.11 mostram os resultados obtidos ao se aplicar os testes de hipóteses nas soluções de cada problema-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. As colunas "GVNS$_r$", "GVNS$_f$" e "GVNS$_{rf}$" apresentam o número de vezes em que a hipótese nula foi rejeitada ao se aplicar os correspondes testes de hipóteses com as disposições GVNS$_r$×Ribeiro *et al.* (2010), GVNS$_f$×Ribeiro *et al.* (2010) e GVNS$_{rf}$×Ribeiro *et al.* (2010), respectivamente.

Tabela 3.11: Número de vezes em que os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ são melhores que o algoritmo de Ribeiro *et al.* (2010).

n	Teste paramétrico			Teste de aleatoriedade		
	GVNS$_r$	**GVNS$_f$**	**GVNS$_{rf}$**	**GVNS$_r$**	**GVNS$_f$**	**GVNS$_{rf}$**
10	0	0	0	0	0	0
20	4	4	4	3	3	3
30	7	11	7	7	9	7
40	15	15	15	14	14	14
50	15	15	15	15	15	15
75	16	16	16	15	16	16
Total	57	61	57	54	57	55

A Tabela 3.10 mostra que os testes de hipóteses paramétrico e de aleatoriedade retornaram resultados semelhantes. O número de vezes em que ambos os testes de hipóteses indicam que as soluções médias dos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ são melhores que as respectivas soluções médias do algoritmo de Ribeiro *et al.* (2010) cresce à medida em que se aumenta o

número de tarefas dos problemas-teste dos conjuntos. Somente para o conjunto de problemas com 10 tarefas que os testes de hipóteses paramétrico e de aleatoriedade não rejeitaram a hipótese nula pelo menos 3 vezes em cada comparação. Por outro lado, para o conjunto de 16 problemas-teste com 75 tarefas, por exemplo, o teste de hipóteses paramétrico indica que as soluções médias dos algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ são melhores que as respectivas soluções médias do algoritmo de Ribeiro *et al.* (2010) para todos os problemas-teste. Para esse mesmo conjunto de problemas-teste, essa mesma conclusão só não foi retornada pelo teste de aleatoriedade na comparação GVNS$_r$×Ribeiro *et al.* (2010), que concluiu que a solução média do algoritmo GVNS$_r$ é melhor que a solução média do algoritmo de Ribeiro *et al.* (2010) para 15 dos 16 problemas-teste.

Uma comparação dos resultados apresentados nas Tabelas 3.10 e 3.11 permite concluir que os testes de hipóteses paramétrico e de aleatoriedade sugerem que os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ são melhores que o melhor algoritmo de Ribeiro *et al.* (2010) um número significativamente maior de vezes que o contrário.

3.4.4.3 Testes estatísticos com o melhor algoritmo de Rosa *et al.* (2017)

Para verificar o número de vezes em que o melhor algoritmo Rosa *et al.* (2017) foi melhor que os algoritmos propostos neste trabalho, foram aplicados os testes de hipóteses paramétrico e de aleatoriedade com os algoritmos dispostos na forma Rosa *et al.* (2017) × GVNS$_r$, Rosa *et al.* (2017) × GVNS$_f$ e Rosa *et al.* (2017) × GVNS$_{rf}$, respectivamente. Com essa disposição, rejeitar a hipótese nula permite afirmar que, com um nível de significância igual a 0,05, há evidências estatísticas de que a solução média obtida pelo algoritmo de Rosa *et al.* (2017) é melhor que a solução média obtida com o respetivo algoritmo na comparação.

As colunas "Teste Paramétrico" e "Teste de Aleatoriedade" da Tabela 3.12 mostram os resultados obtidos ao se aplicar os testes de hipóteses nas soluções de cada problema-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. As colunas "GVNS$_r$", "GVNS$_f$" e "GVNS$_{rf}$" apresentam o número de vezes em que a hipótese nula foi rejeitada ao se aplicar os correspondes testes de hipóteses com as disposições Rosa *et al.* (2017) × GVNS$_r$, Rosa *et al.* (2017) × GVNS$_f$ e Rosa *et al.* (2017) × GVNS$_{rf}$, respectivamente.

Tabela 3.12: Número de vezes em que o melhor algoritmo de Rosa *et al.* (2017) é melhor que os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$.

n	Teste paramétrico			Teste de aleatoriedade		
	GVNS$_r$	**GVNS$_f$**	**GVNS$_{rf}$**	**GVNS$_r$**	**GVNS$_f$**	**GVNS$_{rf}$**
10	0	0	0	0	0	0
20	1	2	1	0	2	0
30	0	1	0	0	1	0
40	0	0	0	0	0	0
50	0	0	0	0	0	0
75	0	0	0	0	0	0
Total	1	3	1	0	3	0

De acordo com a Tabela 3.12, os testes de hipóteses utilizados (teste paramétrico e teste de aleatoriedade) obtiveram resultados similares novamente. Nenhum deles rejeitou a hipótese nula nas comparações associadas a problemas-teste com 10, 40, 50 e 75 tarefas, independentemente do algoritmo comparado o melhor algoritmo de Rosa *et al.* (2017). Na verdade, o teste hipótese paramétrico indica que a solução média obtida pelo algoritmo de Rosa *et al.* (2017) é melhor

que as respectivas soluções médias obtidas com os algoritmos GVNS_r e GVNS_{rf} em apenas um problema-teste de 20 tarefas, e melhor que a solução média do algoritmo GVNS_f em 2 problemas-teste com 20 tarefas e em um problema teste com 30 tarefas. Já o teste de hipóteses de aleatoriedade obteve a mesma conclusão para a comparação Rosa *et al.* (2017) × GVNS_f, mas sugere que a solução média obtida pelo algoritmo de Rosa *et al.* (2017) não é melhor que as respectivas soluções médias dos algoritmos GVNS_r e GVNS_{rf} para nenhum problema-teste considerado.

Por último, foram utilizados os testes de hipóteses paramétrico e de aleatoriedade para se saber o número de vezes em que os algoritmos GVNS_r, GVNS_f e GVNS_{rf} obtiveram soluções médias melhores que as respectivas soluções médias obtidas com o melhor algoritmo de Rosa *et al.* (2017). Com essa finalidade, aplicou-se esses testes com as disposições GVNS_r×Rosa *et al.* (2017), GVNS_f×Rosa *et al.* (2017) e GVNS_{rf}×Rosa *et al.* (2017). Com essas disposições, rejeitar a hipótese nula significa que, com um nível de significância igual a 0,05, há evidências estatísticas suficientes para afirmar que a solução média obtida pelo melhor algoritmo de Rosa *et al.* (2017) é pior que a solução média obtida com o respetivo algoritmo na comparação.

As colunas "Teste Paramétrico" e "Teste de Aleatoriedade" da Tabela 3.13 apresentam os resultados obtidos ao se aplicar os testes de hipóteses paramétrico e de aleatoriedade nas soluções de cada problema-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. As colunas "GVNS_r", "GVNS_f" e "GVNS_{rf}" mostram o número de vezes em que a hipótese nula foi rejeitada ao se aplicar os correspondes testes de hipóteses com as disposições GVNS_r×Rosa *et al.* (2017), GVNS_f×Rosa *et al.* (2017) e GVNS_{rf}×Rosa *et al.* (2017), respectivamente.

Tabela 3.13: Número de vezes em que os algoritmos GVNS_r, GVNS_f e GVNS_{rf} são melhores que o algoritmo de Rosa *et al.* (2017).

n	**Teste paramétrico**			**Teste de aleatoriedade**		
	$\textbf{GVNS}_r$	$\textbf{GVNS}_f$	$\textbf{GVNS}_{rf}$	$\textbf{GVNS}_r$	$\textbf{GVNS}_f$	$\textbf{GVNS}_{rf}$
10	0	0	0	0	0	0
20	0	1	0	0	0	0
30	1	12	1	1	9	1
40	3	12	3	4	12	4
50	11	15	11	11	15	10
75	4	16	6	4	16	6
Total	19	56	21	20	52	21

A Tabela 3.13 mostra que os testes de hipóteses paramétrico e de aleatoriedade retornaram resultados semelhantes mais uma vez. Ambos os testes de hipóteses apontam que o algoritmo GVNS_f foi o que obteve soluções médias melhores que as respectivas soluções médias do algoritmo de Rosa *et al.* (2017) um maior número de vezes. Além disso, de acordo com esses testes de hipóteses, o número de vezes em que a hipótese nula deve ser rejeitada nas comparações GVNS_f×Rosa *et al.* (2017) cresce à medida em que se aumenta o número de tarefas dos problemas-teste dos conjuntos. Para o conjunto de problemas-teste com 75 tarefas, por exemplo, os dois testes de hipóteses indicam que as soluções médias do algoritmo GVNS_f são melhores que as respectivas soluções médias do algoritmo de Ribeiro *et al.* (2010) para todos os 16 problemas-teste. Conforme a Tabela 3.13, os resultados obtidos com as disposições GVNS_r×Rosa *et al.* (2017) e GVNS_{rf}×Rosa *et al.* (2017) são quase os mesmos. As únicas diferenças são observadas nos conjuntos de problemas-teste com 40 e 50 tarefas. Para o conjunto de problemas com 40 tarefas, por exemplo, o teste de hipóteses paramétrico aponta que as soluções médias dos algoritmos GVNS_r e GVNS_{rf} são melhores que as respectivas soluções médias do

algoritmo de Rosa *et al.* (2017) para 3 problemas-teste, ao passo que o teste de hipóteses de aleatoriedade sugere que as soluções médias dos algoritmos GVNS_r e GVNS_{rf} são melhores que as respectivas soluções médias do algoritmo de Rosa *et al.* (2017) para 4 problemas-teste. O maior número de vezes em que os dois testes de hipóteses rejeitam a hipótese nula nas comparações GVNS_r e GVNS_{rf} ocorre no conjunto de problemas-teste com 50 tarefas. Para esse conjunto de problemas, os testes de hipóteses paramétrico e de aleatoriedade afirmam que as soluções médias dos algoritmos GVNS_r e GVNS_{rf} são melhores que as respectivas soluções médias do algoritmo de Rosa *et al.* (2017) um número maior que ou igual a 10 vezes.

Uma comparação dos resultados apresentados nas Tabelas 3.12 e 3.13 possibilita concluir que os testes de hipóteses paramétrico e de aleatoriedade apontam que os algoritmos GVNS_r, GVNS_f e GVNS_{rf} são melhores que o melhor algoritmo de Ribeiro *et al.* (2010) mais vezes que o contrário.

3.4.4.4 Tempos demandados

A Tabela 3.14 apresenta os tempos médios demandados (em segundos) pelos algoritmos GVNS_r, GVNS_f e GVNS_{rf}, bem como pelos melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017) e pelo CPLEX, em cada conjunto de problemas-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste. Exceto para o CPLEX, que foi utilizado para resolver cada problema-teste uma única vez, as médias apresentadas consideram as 30 aplicações dos respectivos algoritmos em cada problema-teste.

Tabela 3.14: Tempos médios (em segundos) demandados pelos melhores algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017), pelos algoritmos GVNS_r, GVNS_f e GVNS_{rf}, bem como pelo CPLEX, para resolver os problemas-teste com até 75 tarefas do SMSPETP-SDS.

n	Ribeiro 2010	Rosa 2017	GVNS_r	GVNS_f	GVNS_{rf}	CPLEX[1]
10	0,42	0,03	0,02	0,02	0,02	524,63
20	4,41	0,42	0,34	0,68	0,34	–
30	27,21	2,31	1,92	7,20	1,93	–
40	99,03	8,61	6,85	40,91	6,88	–
50	289,82	24,69	21,42	159,07	21,55	–
75	2360,57	172,61	155,71	1972,85	162,65	–

[1] O tempo de execução do CPLEX se mostrou proibitivo para resolver os problemas-teste com mais de 10 tarefas. Por esse motivo, esta tabela não contém as informações desses problemas.

Conforme a Tabela 3.14, o CPLEX se mostrou o método mais demorado para resolver os problemas-teste, sendo capaz de resolver somente os problemas com até 10 tarefas dentro do limite de tempo de uma hora de execução. Para todos os conjunto de problemas-teste com a mesma quantidade de tarefas, o algoritmo de Ribeiro *et al.* (2010) exigiu um tempo médio significativamente superior aos respectivos tempos médios demandados pelo algoritmo de Rosa *et al.* (2017) e pelos algoritmos GVNS_r, GVNS_f e GVNS_{rf}. Os tempos médios demandados pelo algoritmo de Rosa *et al.* (2017) e pelos algoritmos GVNS_r, GVNS_f e GVNS_{rf} para resolver os problemas-teste com até 20 tarefas são todos inferiores a um segundo, enquanto o algoritmo de Ribeiro *et al.* (2010) levou 4,41 segundos, em média, para resolver os problemas com 20 tarefas. O algoritmo GVNS_f foi o único, entre os algoritmos propostos neste trabalho, que demandou tempos médios maiores que os respectivos tempos médios demandados pelo algoritmo de Rosa *et al.* (2017). Para os problemas-teste com 75 tarefas, por exemplo, o GVNS_f demandou um tempo médio de 1972,85 segundos, enquanto o algoritmo de Rosa *et al.* (2017) demandou 172,61 segundos. O algoritmo de Rosa *et al.* (2017) e os algoritmos GVNS_r e GVNS_{rf} demandaram

tempos médios próximos para os conjuntos de problemas com o mesmo número de tarefas. No entanto, os tempos médios demandados pelo algoritmo de Rosa *et al.* (2017) são maiores que os respectivos tempos médios demandados pelo algoritmo GVNS$_{rf}$, que, por sua vez, são maiores que os respectivos tempos médios exigidos pelo GVNS$_r$.

3.5 Conclusões do capítulo

Neste capítulo, o SMSPETP foi dividido em dois subproblemas que devem ser resolvidos iterativamente. O primeiro subproblema consiste em determinar a melhor programação de uma dada sequência de tarefas, considerando-se a possibilidade de inserção de ociosidade de máquina. O segundo subproblema é determinar uma sequência de tarefas que, associada a uma dada programação, minimize a soma das penalidades geradas pelas tarefas. O problema de determinar a melhor programação de uma dada sequência de tarefas foi resolvido por algoritmos determinísticos, ao passo que o problema de sequenciamento das tarefas foi resolvido por meio de procedimentos heurísticos. Também propôs-se um método exato para o SMSPETP-SIS, que corresponde ao SMSPETP com tempos de preparação da máquina independentes da sequência de execução das tarefas.

Inicialmente, apresentou-se um algoritmo de programação ótima de uma dada sequência de tarefas da literatura, denotado por OTA (*Optimal Timing Algorithm*). Em seguida, propôs-se um novo algoritmo de programação ótima de uma dada sequência de tarefas, denotado por AAOTO (algoritmo de alocação ótima de tempos ociosos). Propôs-se, também, uma melhoria para o algoritmo OTA, denotada por OTA'. O estudo da complexidade computacional desses algoritmos de programação ótima de uma dada sequência de tarefas do SMSPETP-SDS também é uma contribuição importante deste trabalho. Provou-se que complexidade computacional do algoritmo OTA é $O(n^3)$, enquanto a complexidade computacional dos algoritmos AAOTO e OTA' é $O(n^2)$.

Embora o algoritmo AAOTO seja a principal contribuição deste capítulo, também são contribuições relevantes o desenvolvimento de um algoritmo de enumeração implícita (EI) e de algoritmos baseados na metaheurística VNS (*Variable Neighborhood Search*), denotados por GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$, respectivamente. O algoritmo EI é válido somente para o SMSPETP-SIS e faz uso de resultados teóricos desenvolvidos exclusivamente para essa classe do SMSPETP.

Experimentos computacionais realizados com os problemas-teste apresentados na Seção 2.2 mostram que o algoritmo AAOTO é mais rápido que os algoritmos OTA e OTA'. De fato, apesar de a melhoria proposta para o algoritmo OTA ter possibilitado o GVNS$_f$ resolver os problemas-teste em tempos médios consideravelmente inferiores aos respectivos tempos médios demandados pelo GVNS$_f$ associado ao OTA, os tempos médios demandados pelo algoritmo GVNS$_f$ associado ao AAOTO foram, ainda, significativamente inferiores.

Diante do melhor desempenho apresentado pelo algoritmo AAOTO frente aos algoritmos OTA e OTA' no primeiro conjunto de testes, o algoritmo AAOTO foi utilizado para avaliar cada sequência gerada pelos algoritmos GVNS$_r$, GVNS$_f$, GVNS$_{rf}$ e EI.

O algoritmo EI se mostrou uma boa opção para a resolução exata do SMSPETP-SIS, sendo capaz de resolvê-lo em tempo computacional bem inferior ao tempo requerido pelo CPLEX aplicado à formulação de programação matemática de Gomes Júnior *et al.* (2007). Por outro lado, os algoritmo GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ mostraram-se robustos, encontrando a solução ótima de todos os problemas-teste para os quais essa solução é conhecida, e variabilidade nula em quase todas as execuções para um mesmo problema-teste.

Os algoritmos GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$ foram comparados com a melhor versão do Algoritmo Genético Adaptativo de Ribeiro *et al.* (2010) e com o melhor algoritmo de Rosa *et al.* (2017) em problemas-teste do SMSPETP-SDS. Dois testes de hipóteses, um paramétrico e ou-

tro de aleatoriedade, mostram que os algoritmos propostos encontram soluções médias melhores que as respectivas soluções obtidas com os algoritmos de Ribeiro *et al.* (2010) e de Rosa *et al.* (2017) mais vezes do que o contrário. O algoritmo GVNS$_r$ é o mais rápido entre os algoritmos experimentados. Embora o algoritmo GVNS$_f$ seja o que demanda maior tempo médio entre os algoritmos propostos, ele é o que obtém melhores soluções médias. Além disso, ele requer tempos médios menores que aqueles demandados pelos algoritmos de Ribeiro *et al.* (2010).

Capítulo 4

Formulações matemáticas para o SMSPETP-SDS

Com o objetivo de resolver o SMSPETP-SDS com a garantia de otimalidade da solução encontrada, bem como determinar limites inferiores para esse problema, neste capítulo são apresentadas várias formulações matemáticas para o SMSPETP-SDS. Na Seção 4.1 são apresentadas formulações baseadas em variáveis de precedência, ao passo que na Seção 4.2 são apresentadas formulações baseadas em variáveis de precedência imediata. Na Seção 4.3 é proposta uma formulação matemática baseada em variáveis de posição na sequência para o SMSPETP-SDS, enquanto na Seção 4.4 são apresentadas formulações baseadas em variáveis indexadas no tempo que representam esse problema, bem como cinco novas famílias de restrições válidas para tais formulações. Na Seção 4.7 são propostos algoritmos de separação para as famílias de restrições propostas na Seção 4.4. A Seção 4.8 apresenta e discute os resultados obtidos com as formulações matemáticas e com os algoritmos de separação apresentados neste capítulo. Finalmente, a Seção 4.9 conclui este capítulo.

4.1 Formulações baseadas em variáveis de precedência

Nesta seção é apresentada uma formulação matemática para o SMSPETP-SDS baseada em Manne (1960), denominada formulação VP (formulação baseada em variáveis de precedência). Ademais, é proposto um conjunto de restrições válidas para essa formulação. A formulação VP é descrita na Seção 4.1.1, enquanto o conjunto de restrições válidas para essa formulação é proposto na Seção 4.1.2.

4.1.1 Formulação baseada em Manne (1960) – Formulação VP

A seguir é reproduzida a formulação de programação linear inteira mista de Bustamante (2007) para o SMSPETP-SDS. Esse formulação é baseada na abordagem de Manne (1960), que tratou o problema de *job-shop scheduling*, e será denotada por VP (formulação baseada em variáveis de precedência).

Para cada tarefa $x \in I$, a data de início de execução, a antecipação e o atraso de x são representados por s_x, e_x e t_x, respectivamente. Sejam w_{xy} variáveis que determinam a relação de precedência das tarefas x e y, de modo que $w_{xy} = 1$, se a tarefa y for executada depois da tarefa x, e $w_{xy} = 0$, caso contrário, $\forall\ x, y \in I \quad \text{e} \quad x \neq y$. Se M_{xy} é uma constante de valor suficientemente grande, $\forall\ x, y \in I \quad \text{e} \quad x \neq y$, a formulação VP é dada por:

$$\text{(VP)} \qquad \min \sum_{x \in I} (\alpha_x e_x + \beta_x t_x) \tag{4.1}$$

$$\text{s. a} \quad s_x + P_x + w_{xy}(M_{xy} + S_{xy}) - M_{xy} \leq s_y \quad \forall\, x, y \in I \ \text{ e } \ x \neq y \tag{4.2}$$

$$w_{xy} + w_{yx} = 1 \quad \forall\, x, y \in I \ \text{ e } \ x \neq y \tag{4.3}$$

$$s_x + P_x + e_x \geq E_x \quad \forall\, x \in I \tag{4.4}$$

$$s_x + P_x - t_x \leq T_x \quad \forall\, x \in I \tag{4.5}$$

$$s_x \geq 0 \quad \forall\, x \in I \tag{4.6}$$

$$e_x \geq 0 \quad \forall\, x \in I \tag{4.7}$$

$$t_x \geq 0 \quad \forall\, x \in I \tag{4.8}$$

$$w_{xy} \in \{0, 1\} \quad \forall\, x, y \in I \ \text{ e } \ x \neq y \tag{4.9}$$

O objetivo da formulação VP é minimizar o somatório (4.1), ou seja, minimizar a soma ponderada das antecipações e atrasos das tarefas. As desigualdades (4.2) garantem que existe tempo suficiente para executar uma tarefa x e preparar a máquina antes do início da execução da tarefa seguinte y, ou seja, elas garantem que a execução da tarefa y não será iniciada antes do término da execução da tarefa x. As restrições (4.3) garantem que cada tarefa será executada exatamente uma vez. As desigualdades (4.4) e (4.5) determinam as antecipações e os atrasos das tarefas de acordo com as respectivas janelas de conclusão. As restrições (4.6), (4.7), (4.8) e (4.9) dizem respeito aos domínios das variáveis.

O número de variáveis de decisão na formulação VP (restrições (4.6)–(4.9)) é $3n+n(n-1) = n^2+2n$, enquanto o número total de restrições (restrições (4.2)–(4.5)) é $2n(n-1)+2n = 2n^2$. Dados $x, y \in I$ e $x \neq y$, se $w_{xy} = 1$, então a constante M_{xy} não interfere na respectiva restrição (4.2). Caso contrário, isto é, se $w_{xy} = 0$, a restrição que contém M_{xy} é reduzida a $M_{xy} \geq s_x + P_x - s_y$. Logo, se s_x^{UB} é um limite superior para s_x e s_y^{LB} é um limite inferior para s_y, então pode-se utilizar $M_{xy} = s_x^{UB} + P_x - s_y^{LB}$.

4.1.2 Novas restrições válidas para a formulação VP – Formulação VP'

O conjunto de restrições (4.10) foi proposto por Nemhauser & Savelsbergh (1992) para uma classe de problemas de programação de tarefas com tempos de preparação da máquina independentes da sequência de execução das tarefas, para a qual não é vantajosa a ociosidade de máquina. Esse conjunto de restrições garante que a execução de uma dada tarefa $x \in I$ somente seja iniciada após a conclusão de todas as tarefas executadas antes de x.

$$\sum_{y \in I \setminus \{x\}} P_y \cdot w_{yx} \leq \quad s_x \quad \forall\, x \in I \tag{4.10}$$

Embora sejam válidas também para o SMSPETP-SDS, as restrições (4.10) não consideram os tempos de preparação da máquina entre a execução de duas tarefas. A Proposição 4.1 generaliza o conjunto de restrições (4.10) a fim de considerar esses tempos de preparação da máquina.

Proposição 4.1 *Para toda tarefa $x \in I$, tem-se:*

$$s_x \geq \sum_{y \in I \setminus \{x\}} \left(P_y + \min_{z \in I \setminus \{y\}} S_{yz} \right) \cdot w_{yx}. \tag{4.11}$$

As restrições (4.11) dizem que, dada uma tarefa x, além de tempo suficiente para executar todas as tarefas programadas antes de x, deve haver um tempo mínimo para realizar as respectivas preparações da máquina antes de iniciar a execução de x. Logo, o conjunto de restrições (4.11) é mais forte que o conjunto (4.10), no sentido de proibir um maior número de soluções inviáveis para o SMSPETP-SDS.

A formulação VP acrescida das n restrições (4.11) será denotada por VP'.

4.2 Formulações baseadas em variáveis de precedência imediata

Esta seção apresenta duas formulações matemática para o SMSPETP-SDS. A primeira delas, descrita na Seção 4.2.1, é baseada em Miller *et al.* (1960) e será denominada VPI (formulação baseada em variáveis de precedência imediata). A outra formulação, descrita na Seção 4.2.2, além de usar variáveis de precedência imediata, introduz um conjunto alternativo de variáveis de decisão para representar as datas de início de execução as tarefas.

4.2.1 Formulação baseada em Miller *et al.* (1960) – Formulação VPI

Devido às similaridades entre o problema do caixeiro viajante (TSP – *Travelling Salesman Problem*) e o problema de programação de tarefas (Baker & Keller, 2010), é possível utilizar as variáveis de precedência imediata, propostas inicialmente por Miller *et al.* (1960) para modelar o TSP, para construir uma formulação de programação inteira mista para o SMSPETP-SDS. A formulação apresentada nesta seção, denominada VPI (formulação baseada em restrições de precedência imediata), é baseada nos trabalhos de Miller *et al.* (1960), Gomes Júnior *et al.* (2007) e Nogueira *et al.* (2014)

Para auxiliar na modelagem, é utilizada uma tarefa fictícia, denominada x_0. Essa tarefa x_0 deve ser executada duas vezes, de modo que é a primeira e a última tarefa a ser executada.

Sejam u_{xy} variáveis de decisão que determinam a sequência em que as tarefas serão executadas, de modo que $u_{xy} = 1$, se a tarefa y for executada imediatamente após a tarefa x, e $u_{xy} = 0$, caso contrário, $\forall\, x,\, y \in I \cup \{x_0\}$. Considera-se que $P_{x_0} = S_{x_0,x} = S_{x,x_0} = 0,\, \forall\, x \in I$. Assim como na formulação VP, s_x são variáveis de decisão que determinam as datas de início de execução das tarefas $x \in I \cup \{x_0\}$, ao passo que e_x e t_x são variáveis de decisão que determinam os tempos de antecipação e de atraso, respectivamente, das tarefas $x \in I$. Se M_{xy} são valores suficientemente grandes, $\forall x,\, y \in I \cup \{x_0\}$ e $x \neq y$, a formulação VPI é dada por:

$$\text{(VPI)} \qquad \min \sum_{x \in I} (\alpha_x e_x + \beta_x t_x) \tag{4.12}$$

$$\text{s. a.} \qquad \sum_{\substack{y \in I \cup \{x_0\} \\ y \neq x}} u_{xy} = 1 \qquad \forall\, x \in I \cup \{x_0\} \tag{4.13}$$

$$\sum_{\substack{x \in I \cup \{x_0\} \\ x \neq y}} u_{xy} = 1 \qquad \forall\, y \in I \cup \{x_0\} \tag{4.14}$$

$$s_x + P_x + S_{xy} u_{xy} + M_{xy}(u_{xy} - 1) \leq s_y \qquad \forall\, x,\, y \in I \cup \{x_0\} \text{ e } x \neq y \tag{4.15}$$

$$s_x + P_x + e_x \geq E_x \qquad \forall\, x \in I \tag{4.16}$$

$$s_x + P_x - t_x \leq T_x \qquad \forall\, x \in I \tag{4.17}$$

$$s_x \geq 0 \qquad \forall\, x \in I \cup \{x_0\} \tag{4.18}$$

$$e_x \geq 0 \qquad \forall\, x \in I \tag{4.19}$$

$$t_x \geq 0 \qquad \forall\, x \in I \tag{4.20}$$

$$u_{xy} \in \{0, 1\} \qquad \forall\, x, y \in I \cup \{x_0\} \text{ e } x \neq y \tag{4.21}$$

A função objetivo, representada pela expressão (4.12), busca a minimização da soma ponderada das antecipações e atrasos. As restrições (4.13) e (4.14) garantem que cada tarefa terá apenas uma tarefa imediatamente sucessora e uma tarefa imediatamente antecessora, respectivamente. As restrições (4.15) garantem que existe tempo suficiente para executar uma tarefa antes de iniciar a execução da tarefa seguinte. As restrições (4.16) e (4.17) determinam as antecipações e os atrasos de acordo com as janelas de conclusão das tarefa. As restrições (4.18), (4.19), (4.20) e (4.21) dizem respeito aos domínios das variáveis de decisão.

O número de variáveis de decisão na formulação VPI (restrições (4.18)–(4.21)) é $(n+1) + 2n + (n+1)n = n^2 + 4n + 1$, enquanto o número total de restrições (restrições (4.13)–(4.6)) é $2(n+1) + (n+1)n + 2n = n^2 + 5n + 2$.

Gomes Júnior *et al.* (2007) consideram $M_{xy} = 1000,\ \forall\, x, y \in I \cup \{x_0\}$ e $x \neq y$. No entanto, esse parâmetro da formulação VPI pode ser dependente do problema. Dados $x,\, y \in I \cup \{x_0\}$ e $x \neq y$, se $u_{xy} = 1$, então a constante M_{xy} não interfere na respectiva restrição (4.15). Caso contrário, isto é, se $u_{xy} = 0$, a restrição que contém M_{xy} é reduzida a $M_{xy} \geq s_x + P_x - s_y$. Logo, se s_x^{UB} é um limite superior para s_x e s_y^{LB} é um limite inferior para s_y, então pode-se utilizar $M_{xy} = s_x^{UB} + P_x - s_y^{LB}$.

4.2.2 Formulação alternativa – Formulação VPIA

Uma alternativa para eliminar as constantes M_{xy} presentes na formulação VPI é redefinir as variáveis s_x como variáveis s_{xy}, de modo que $u_{xy} = 0$ implique em $s_{xy} = 0$ (Ascheuer *et al.*, 2001). As variáveis s_{xy} foram propostas por van Eijl (1995) e Maffioli & Sciomachen (1997) (o primeiro autor propôs tais variáveis para modelar o TSP, enquanto os outros dois autores as introduziu para modelar uma classe de problemas de programação de tarefas). A seguir é proposto uma nova formulação matemática para o SMSPETP-SDS, denominada VPIA (formulação VPI alternativa), a qual foi inspirada nos trabalhos de van Eijl (1995), Maffioli & Sciomachen (1997) e Nogueira *et al.* (2014).

Sejam s_{xy} variáveis de decisão tais que, $\forall x,\, y \in I \cup \{x_0\}$ e $x \neq y$:

$$s_{xy} = \begin{cases} s_x, & \text{se } u_{xy} = 1; \\ 0, & \text{caso contrário.} \end{cases}$$

Em outras palavras, se $u_{xy} = 1$, então s_{xy} representa a data de início de execução da tarefa x e que a tarefa y é executada imediatamente após a tarefa x. Se s_x^{UB} representa um limite superior para a data de início da tarefa x, $\forall x \in I \cup \{x_0\}$, então a formulação VPIA é dada por:

$$\text{(VPIA)} \qquad \min \sum_{x \in I} (\alpha_x e_x + \beta_x t_x) \tag{4.22}$$

$$\text{s. a.} \qquad \sum_{\substack{y \in I \cup \{x_0\} \\ y \neq x}} u_{xy} = 1 \qquad \forall\, x \in I \cup \{x_0\} \tag{4.23}$$

$$\sum_{\substack{x \in I \cup \{x_0\} \\ x \neq y}} u_{xy} = 1 \qquad \forall\, y \in I \cup \{x_0\} \tag{4.24}$$

$$\sum_{\substack{x \in I \\ x \neq y}} \big(s_{xy} + (P_x + S_{xy}) u_{xy}\big) \leq \sum_{\substack{x \in I \cup \{x_0\} \\ x \neq y}} s_{yx} \qquad \forall\, y \in I \tag{4.25}$$

$$s_{xy} \leq s_x^{UB} \cdot u_{xy} \qquad \forall\, x, y \in I \cup \{x_0\} \text{ e } x \neq y \tag{4.26}$$

$$\sum_{\substack{y \in I \cup \{x_0\} \\ y \neq x}} s_{xy} + P_x + e_x \geq E_x \qquad \forall\, x \in I \tag{4.27}$$

$$\sum_{\substack{y \in I \cup \{x_0\} \\ y \neq x}} s_{xy} + P_x - t_x \leq T_x \qquad \forall\, x \in I \tag{4.28}$$

$$e_x \geq 0 \qquad \forall\, x \in I \tag{4.29}$$

$$t_x \geq 0 \qquad \forall\, x \in I \tag{4.30}$$

$$u_{xy} \in \{0, 1\} \qquad \forall\, x, y \in I \cup \{x_0\} \text{ e } x \neq y \tag{4.31}$$

$$s_{xy} \geq 0 \qquad \forall\, x, y \in I \cup \{x_0\} \text{ e } x \neq y \tag{4.32}$$

As restrições (4.23), (4.24), (4.29), (4.30) e (4.31) são oriundas da formulação VPI. As restrições (4.25) e (4.26) garantem que existe tempo suficiente para executar uma tarefa antes de iniciar a execução da tarefa seguinte, ou seja, elas substituem as restrições (4.15) da formulação VPI. As restrições (4.27) e (4.28) determinam as antecipações e os atrasos, de acordo com a respectiva janela de conclusão de cada tarefa. As restrições (4.32) dizem respeito ao domínio das variáveis de decisão s_{xy}.

O número de variáveis de decisão na formulação VPIA (restrições (4.29)–(4.32)) é $2n + 2(n+1)n = 2n^2 + 4n$, ao passo que o número total de restrições (restrições (4.23)–(4.28)) é $2(n+1) + n + (n+1)n + 2n = n^2 + 6n + 2$.

4.3 Formulação baseada em variáveis de posição na sequência – Formulação VPS

Nesta seção é proposta uma nova formulação inteira mista para representar o SMSPETP-SDS, a qual é inspirada nos trabalhos de Wagner (1959), que propôs variáveis baseadas na posição de cada tarefa na sequência de execução para modelar uma classe de problemas de *job-shop scheduling*, e Bustamante (2007), que adaptou a formulação de Wagner (1959) para o SMSPETP-SDS com penalidades por unidade de tempo de antecipação e atraso independentes das tarefas.

Sejam s_x, e_x e t_x variáveis de decisão tais como na formulação VP e sejam v_{xi} e w_{xyi} variáveis de decisão tais que, $\forall\, x, y \in I$, $x \neq y$ e $\forall\, i \in \{1, 2, \ldots, n\}$:

- $v_{xi} = \begin{cases} 1, & \text{se } x \text{ é a } i\text{-ésima tarefa a ser executada;} \\ 0, & \text{caso contrário.} \end{cases}$
- $w_{xyi} = \begin{cases} 1, & \text{se } x \text{ é a } i\text{-ésima tarefa e } y \text{ é a } (i{+}1)\text{-ésima tarefa a serem executadas;} \\ 0, & \text{caso contrário.} \end{cases}$

A formulação matemática dada pelas equações (4.33)–(4.44) representa o SMSPETP-SDS.

$$\text{(VPS)} \qquad \min \sum_{x \in I} (\alpha_x e_x + \beta_x t_x) \tag{4.33}$$

$$\text{s.a.} \quad \sum_{x \in I} v_{xi} = 1 \qquad \forall\, i \in \{1, 2, \ldots, n\} \tag{4.34}$$

$$\sum_{i=1}^{n} v_{xi} = 1 \qquad \forall\, x \in I \tag{4.35}$$

$$v_{xi} + v_{y,i+1} - w_{xyi} \leq 1 \qquad \forall\, x, y \in I,\ x \neq y \ \text{ e } \ \forall\, i \in \{1, 2, \ldots, n-1\} \tag{4.36}$$

$$\sum_{x \in I} \left(s_x v_{x,i+1} - (s_x + P_x) v_{xi} - \sum_{y \in I \setminus \{x\}} S_{xy} w_{xyi} \right) \geq 0 \qquad \forall\, i \in \{1, 2, \ldots, n-1\} \tag{4.37}$$

$$s_x + P_x + e_x \geq E_x \qquad \forall\, x \in I \tag{4.38}$$

$$s_x + P_x - t_x \leq T_x \qquad \forall\, x \in I \tag{4.39}$$

$$s_x \geq 0 \qquad \forall\, x \in I \tag{4.40}$$

$$e_x \geq 0 \qquad \forall\, x \in I \tag{4.41}$$

$$t_x \geq 0 \qquad \forall\, x \in I \tag{4.42}$$

$$v_{xi} \in \{0, 1\} \qquad \forall\, x \in I \ \text{ e } \ \forall\, i \in \{1, 2, \ldots, n\} \tag{4.43}$$

$$w_{xyi} \in \{0, 1\} \qquad \forall\, x, y \in I,\ x \neq y \ \text{ e } \ \forall\, i \in \{1, 2, \ldots, n\} \tag{4.44}$$

A expressão (4.33) representa a função objetivo do problema. As restrições (4.34) e (4.35) garantem que cada tarefa será executada exatamente uma vez. As restrições (4.36) e (4.37) asseguram que existe tempo suficiente para executar uma tarefa antes de iniciar a execução da tarefa sucessora. As restrições (4.38) e (4.39) determinam a antecipação e o atraso de cada tarefa. As restrições (4.40)–(4.44) definem os domínios das variáveis de decisão.

Embora as restrições (4.37) sejam não-lineares, é possível linearizá-las. Para isso, considere a variável auxiliar q_{xi} tal que $q_{xi} = s_x v_{xi},\ \forall\, x \in I$ e $\forall\, i \in \{1, 2, \ldots, n\}$. Para toda tarefa $x \in I$, se s_x^{UB} é um limite superior para s_x, então $0 \leq q_{xi} \leq s_x$ e $s_x + s_x^{UB}(v_{xi} - 1) \leq q_{xi} \leq s_x^{UB} v_{xi}$.

Uma nova formulação de programação linear inteira mista para o SMSPETP-SDS, denotada por VPS (formulação baseada em variáveis de posição na sequência), é obtida ao substituir as restrições (4.37) pelas restrições (4.45)–(4.49).

$$\sum_{x\in I}\left(q_{x,i+1}-q_{xi}-P_x v_{xi}-\sum_{y\in I\setminus\{x\}} S_{xy}w_{xyi}\right) \geq 0 \quad \forall\, i\in\{1,\,2,\,\ldots,\,n-1\} \tag{4.45}$$

$$q_{xi}-s_x \leq 0 \quad \forall\, x\in I \quad \text{e} \quad \forall i\in\{1,\,2,\,\ldots,\,n\} \tag{4.46}$$

$$q_{xi}-s_x^{UB}v_{xi} \leq 0 \quad \forall\, x\in I \quad \text{e} \quad \forall i\in\{1,\,2,\,\ldots,\,n\} \tag{4.47}$$

$$s_x+s_x^{UB}v_{xi}-q_{xi} \leq s_x^{UB} \quad \forall\, x\in I \quad \text{e} \quad \forall i\in\{1,\,2,\,\ldots,\,n\} \tag{4.48}$$

$$q_{xi} \geq 0 \quad \forall\, x\in I \quad \text{e} \quad \forall i\in\{1,\,2,\,\ldots,\,n\} \tag{4.49}$$

O número de variáveis de decisão na formulação VPS (restrições (4.40)–(4.44) e restrições (4.49)) é $3n+n^2+n^2(n-1)+n^2=n^3+n^2+3n$, ao passo que o número total de restrições (restrições (4.34)–(4.36), restrições (4.38)–(4.39) e restrições (4.45)–(4.48)) é $2n+n(n-1)^2+2n+(n-1)+3n^2=n^3+n^2+6n-1$.

4.4 Formulações baseadas em variáveis indexadas no tempo

Nesta seção são apresentadas formulações baseadas em variáveis indexadas no tempo para o SMSPETP-SDS. Tal abordagem foi proposta por Sousa & Wolsey (1992) para modelar uma classe de problemas de programação de tarefas em uma máquina.

Seja $H_x=\left\{s_x^{LB},\,s_x^{LB}+1,\,\ldots,s_x^{UB}\right\}$ o conjunto das possíveis datas de início de execução da tarefa $x\in I$. Sejam l_{xh} variáveis de decisão tais que, $\forall\, x\in I$ e $\forall\, h\in H_x$,

$$l_{xh}=\begin{cases}1, & \text{se a execução da tarefa } x \text{ inicia na data } h;\\ 0, & \text{caso contrário.}\end{cases}$$

No restante deste trabalho, $\lfloor\lambda\rfloor_x$ representa $\max(s_x^{LB},\lambda)$ e $\lceil\lambda\rceil_x$ representa $\min(\lambda,s_x^{UB})$, para toda tarefa $x\in I$ e para todo número real λ.

A formulação inteira mista de Rosa & Souza (2009) é apresentada na Seção 4.4.1, enquanto uma nova formulação inteira mista baseada em Nogueira *et al.* (2014) é proposta na Seção 4.4.2. Essas duas formulações inteiras mistas são reduzidas a duas formulações binárias na Seção 4.4.3. Duas novas formulações binárias para o SMSPETP-SDS são propostas na Seção 4.4.4, ao passo que a Seção 4.6 propõe cinco famílias de restrições válidas para as formulações indexadas no tempo que representam o SMSPETP-SDS.

4.4.1 Formulação linear inteira mista de Rosa & Souza (2009) – Formulação IMR

Nesta seção é apresentada a formulação linear inteira mista proposta por (Rosa & Souza, 2009) para o SMSPETP-SDS, denotada por IMR. Sejam e_x e t_x variáveis de decisão tais como na formulação VP (vide Seção 4.1). A formulação IMR é dada pelas equações (4.50)–(4.57).

$$\text{(IMR)} \qquad \min \sum_{x \in I} (\alpha_x e_x + \beta_x t_x) \tag{4.50}$$

$$\text{s.a.} \qquad l_{xh} + \sum_{k=\lfloor h \rfloor_y}^{\lceil h+P_x+S_{xy}-1 \rceil_y} l_{yk} \leq 1 \qquad \forall\, x, y \in I,\ x \neq y \ \text{ e } \ \forall\, h \in H_x \tag{4.51}$$

$$\sum_{h \in H_x} l_{xh} = 1 \qquad \forall\, x \in I \tag{4.52}$$

$$\sum_{h \in H_x} h \cdot l_{xh} + P_x + e_x \geq E_x \qquad \forall\, x \in I \tag{4.53}$$

$$\sum_{h \in H_x} h \cdot l_{xh} + P_x - t_x \leq T_x \qquad \forall\, x \in I \tag{4.54}$$

$$e_x \geq 0 \qquad \forall\, x \in I \tag{4.55}$$

$$t_x \geq 0 \qquad \forall\, x \in I \tag{4.56}$$

$$l_{xh} \in \{0, 1\} \qquad \forall\, x \in I \ \text{ e } \ \forall\, h \in H_x \tag{4.57}$$

A função objetivo, representada pela equação (4.50), busca a minimização da soma ponderada das antecipações e atrasos. As restrições (4.51) garantem que existe tempo suficiente para executar uma tarefa e preparar a máquina antes de iniciar a execução da tarefa seguinte. As restrições (4.52) garantem que cada tarefa será executada uma única vez. As restrições (4.53) e (4.54) determinam as antecipações e os atrasos das tarefas. As restrições (4.55), (4.56) e (4.57) dizem respeito aos domínios das variáveis de decisão.

O número de variáveis de decisão da formulação IMR é dado por $2n + \sum_{x \in I} |H_x|$, em que $|H_x|$ representa a cardinalidade do conjunto H_x. Já o número de restrições é dado por $3n + (n-1) \sum_{x \in I} |H_x|$. Se $H_x = H,\ \forall x \in I$, então o número de variáveis da formulação IMR é dado por $n(2 + |H|)$ e o número de restrições é dado por $n\big(3 + (n-1)|H|\big)$.

4.4.2 Formulação baseada em Nogueira *et al.* (2014) – Formulação IMN

As restrições (4.51), presentes na formulação IMR, podem ser substituídas pelas restrições (4.58), apresentadas em Nogueira *et al.* (2014) para uma classe de problemas de programação de tarefas com tempos de preparação dependentes da sequência de execução das tarefas.

$$l_{xh} + \sum_{k=\lfloor h-P_y-S_{yx}+1 \rfloor_y}^{\lceil h+P_x+S_{xy}-1 \rceil_y} l_{yk} \leq 1, \quad \forall\, x, y \in I,\ x \neq y,\ h \in H_x \tag{4.58}$$

As restrições (4.58) asseguram que, se a execução da tarefa $x \in I$ é iniciado na data $h \in H_x$, então nenhuma outra tarefa $y \in I \setminus \{x\}$ pode ser iniciada no intervalo $\{h - P_y - S_{yx} + 1, \ldots, h + P_x + S_{xy} - 1\}$. Em outras palavras, as restrições (4.58) garantem que não haverá duas tarefas sendo executadas simultaneamente. Observa-se que toda restrição do conjunto de restrições (4.51) é dominada por uma restrição pertencente ao conjunto de restrições (4.58) no sentido de, ainda que removida a condição de integralidade das variáveis de decisão, toda solução não-negativa de uma restrição pertencente a (4.51) é também solução de uma restrição pertencente a (4.58).

A formulação obtida da formulação IMR por meio da substituição das restrições (4.51) pelas restrições (4.58) é denotada por IMN. Os números de variáveis de decisão e de restrições da formulação IMN são iguais aos da formulação IMR.

4.4.3 Formulações binárias – Formulações BR e BN

As variáveis de decisão não binárias (variáveis e_x e t_x) das formulações IMR e IMN podem ser eliminadas dessas formulações. Para isso, assim como em (Tanaka, 2012), $\forall x \in I$, considere a função custo $g_x : H_x \to \mathbb{R}$ dada por:

$$g_x(h) = \alpha_x \cdot \max(E_x - h - P_x, 0) + \beta_x \cdot \max(h + P_x - T_x, 0). \tag{4.59}$$

O valor $g_x(h)$ corresponde à penalidade associada à tarefa x caso sua execução seja iniciada na data h. Logo, a função objetivo das formulações IMR e IMN, dada pela equação (4.50), pode ser substituída pela equação (4.60). Essa substituição torna as variáveis e_x e t_x, $\forall x \in I$, bem como as restrições (4.53)–(4.56), desnecessárias.

$$\min \quad \sum_{x \in I} \sum_{h \in H_x} g_x(h) \cdot l_{xh} \tag{4.60}$$

As formulações obtidas das formulações IMR e IMN por meio da eliminação das variáveis e_x e t_x são denotadas por BR e BN, respectivamente. O número de variáveis de decisão e o número de restrições dessas formulações são dados por $\sum_{x \in I} |H_x|$ e $n + (n-1)\sum_{x \in I} |H_x|$, respectivamente. Se $H_x = H$, $\forall x \in I$, então o número de variáveis das formulações BR e BN é dado por $n|H|$ e o número de restrições dessas formulações é dado por $n\big(1 + (n-1)|H|\big)$.

4.4.4 Novas formulações binárias – Formulações B1 e B2

Uma desvantagem das formulações indexadas no tempo é o elevado número de variáveis de decisão e de restrições, uma vez que tanto o número de variáveis quanto o número de restrições é dependente do horizonte de planejamento das tarefas. Nesta seção são propostas duas novas formulações binárias para o SMSPETP-SDS, denominadas B1 e B2, que possuem um número de restrições significativamente inferior ao das formulações BR e BN.

As restrições (4.61) foram definidas por Sousa & Wolsey (1992) para o problema de programação de tarefas sem tempos de preparação da máquina. Elas asseguram que a máquina execute no máximo uma tarefa por vez.

$$\sum_{x \in I} \sum_{k=\lfloor h - P_x + 1 \rfloor_x}^{\lceil h \rceil_x} l_{xk} \quad \leq \quad 1, \quad \forall\, h \in \bigcup_{x \in I} H_x \tag{4.61}$$

Embora as restrições (4.61) sejam válidas também para o problema de programação de tarefas com tempos de preparação da máquina dependentes da sequência execução, Nogueira *et al.* (2014) propõem uma generalização dessas restrições para tal classe de problemas. O conjunto de restrições (4.62), proposto por Nogueira *et al.* (2014), é mais forte que o conjunto (4.61) no sentido de eliminar um maior número de soluções inviáveis para o SMSPETP-SDS.

$$\sum_{x \in I} \sum_{k=\lfloor h - P_x - \min\limits_{y \in I \setminus \{x\}} S_{xy} + 1 \rfloor_x}^{\lceil h \rceil_x} l_{xk} \quad \leq \quad 1, \quad \forall\, h \in \bigcup_{x \in I} H_x \tag{4.62}$$

Resguardando as diferenças nas notações utilizadas e no problema tratado, Avella *et al.* (2016) expandiram a família de restrições (4.62) para a família (4.63).

$$\sum_{x\in I'}\sum_{k=\lfloor h-P_x-\min\limits_{y\in I'\setminus\{x\}} S_{xy}+1\rfloor_x}^{\lceil h\rceil_x} l_{xk} \quad \leq \quad 1, \quad \forall\, I'\subseteq I \ \text{ e } \ \forall\, h\in \bigcup_{x\in I'} H_x \tag{4.63}$$

As formulações BR e BN são compostas de uma família de restrições que garante que toda tarefa seja executada (restrições (4.52) e um conjunto de restrições que proíbe a máquina de executar mais de uma tarefa simultaneamente (restrições (4.51) na formulação BR e restrições (4.58) na formulação BN). O número de restrições da família (4.52) depende apenas do número de tarefas a serem executadas. Por outro lado, a cardinalidade das famílias de restrições (4.51) e (4.58) depende, também, do horizonte de planejamento das tarefas em I.

A fim de construir formulações indexadas no tempo mais compactas que a formulação BR, as restrições (4.51) podem ser substituídas por uma das duas seguintes famílias de restrições:

$$\sum_{k=\lfloor h-P_x-S_{xy}+1\rfloor_x}^{\lceil h\rceil_x} l_{xk} + \sum_{k=\lfloor h-P_y-S_{yx}+1\rfloor_y}^{\lceil h\rceil_y} l_{yk} \quad \leq \quad 1 \quad \forall\, x,y\in I,\ x\neq y,\ h\in H_x\cup H_y \tag{4.64}$$

ou

$$\sum_{k=\lfloor h\rfloor_x}^{\lceil h+P_y+S_{yx}-1\rceil_x} l_{xk} + \sum_{k=\lfloor h\rfloor_y}^{\lceil h+P_x+S_{xy}-1\rceil_y} l_{yk} \quad \leq \quad 1 \quad \forall\, x,y\in I,\ x\neq y,\ h\in H_x\cup H_y. \tag{4.65}$$

A família de restrições (4.64) é a subfamília da família (4.63) obtida ao se considerar somente os subconjuntos $I'\subset I$ tais que $|I'|=2$. Assim como a família (4.64), a família (4.65) também proíbe que duas tarefas sejam executadas simultaneamente. Ademais, ainda que retirada a condição de integralidade das variáveis de decisão, toda solução não-negativa de uma restrição pertencente ao conjunto de restrições (4.51) é também solução de uma restrição pertencente ao conjunto de restrições (4.65), isto é, toda restrição de (4.51) é dominada por uma restrição de (4.65).

A formulação binária obtida da formulação BR ao substituir as restrições (4.51) pelas restrições (4.64) será denotada por B1. Por outro lado, a formulação obtida de BR pela substituição das restrições (4.51) pelas restrições (4.65) será denotada por B2. Se $H_x = H,\ \forall x\in I$, então o número de restrições das famílias (4.64) e (4.65) é igual à metade do número de restrições da família (4.51), isto é, $n\big(1+(n-1)|H|/2\big)$ restrições. Logo, embora as formulações B1, B2, BR e BN tenham o mesmo número de variáveis de decisão, as formulações B1 e B2 possuem um menor número de restrições que as outras duas.

4.5 Principais características das formulações matemáticas

A Tabela 4.1 resume as principais características das formulações de programação matemática para o SMSPETP-SDS apresentadas nas Seções 4.1–4.4. Nessa tabela, a primeira coluna indica as formulações matemáticas apresentadas, a segunda e terceira colunas apresentam os números de variáveis de decisão binárias e não binárias, respectivamente, contidas em cada formulação, ao passo que a última coluna apresenta o número de restrições em cada formulação.

De acordo com a Tabela 4.1, o número de variáveis binárias nas formulações VP, VP', VPI e VPIA é $O(n^2)$, na formulação VPS é n^3 e nas formulações baseadas em variáveis indexadas no tempo é $n|H|$. As formulações BR, BN, B1 e B2 só possuem variáveis binárias, enquanto

Tabela 4.1: Número de variáveis e de restrições nas formulações matemáticas para o SMSPETP-SDS apresentadas nas Seções 4.1–4.4.

Formulação	Variáveis de decisão		Restrições
	Binárias	Não binárias	
VP	$n(n-1)$	$3n$	$2n^2$
VP'	$n(n-1)$	$3n$	$2n^2+n$
VPI	$n(n+1)$	$3n+1$	n^2+5n+2
VPIA	$n(n+1)$	$n(n+3)$	n^2+6n+2
VPS	n^3	$n(n+3)$	n^3+n^2+6n-1
IMR	$n\|H\|$	$2n$	$3n+(n^2-n)\|H\|$
IMN	$n\|H\|$	$2n$	$3n+(n^2-n)\|H\|$
BR	$n\|H\|$	0	$n+(n^2-n)\|H\|$
BN	$n\|H\|$	0	$n+(n^2-n)\|H\|$
B1	$n\|H\|$	0	$n+(n^2-n)\|H\|/2$
B2	$n\|H\|$	0	$n+(n^2-n)\|H\|/2$

o número de variáveis não binárias nas formulações VP, VP', VPI, IMN e IMR é $O(n)$ e nas formulações VPIA e VPS é $O(n^2)$. Ainda conforme a Tabela 4.1, o número de restrições nas formulações VP, VP', VPI e VPIA é $O(n^2)$, na formulação VPS é n^3 e nas formulações indexadas no tempo é $n^2|H|$. Ademais, o número de restrições nas formulações B1 e B2 são aproximadamente iguais à metade do número de restrições nas demais formulações indexadas no tempo.

4.6 Novas restrições válidas para as formulações indexadas no tempo

Esta seção propõe novas famílias de restrições válidas para as formulações indexadas no tempo de problemas de programação de tarefas com tempos de preparação da máquina dependentes da sequência de execução. Essas famílias foram desenvolvidas a fim de proporcionar a essas formulações uma relaxação linear com soluções mais próximas do problema inteiro.

A primeira família de restrições válidas proposta é inspirada em Sousa & Wolsey (1992). Nesse trabalho, os autores propõem o conjunto de restrições (4.66) para formulações indexadas no tempo de problemas de programação de tarefas sem tempos de preparação da máquina entre a execução de duas tarefas.

$$\sum_{k=\lfloor h-P_x+1\rfloor_x}^{\lceil h+\Delta-1\rceil_x} l_{xk} + \sum_{y\in I\setminus\{x\}:\, P_y\geq\Delta} \sum_{k=\lfloor h-P_y+\Delta\rfloor_y}^{\lceil h\rceil_y} l_{yk} \leq 1 \quad \forall x\in I,\ h\in \bigcup_{y\in I\setminus\{x\}} H_y \text{ e } \Delta\in\{2,3,\ldots,\max_{y\in I\setminus\{x\}} P_y\} \tag{4.66}$$

A Proposição 4.2 generaliza as restrições (4.66) para problemas de programação de tarefas com tempos de preparação da máquina dependentes da sequência de execução. A família de restrições dada por tal proposição será denominada "Família 1".

Proposição 4.2 (Família 1) *Seja $I'\subseteq I$ um subconjunto de tarefas tal que $|I'|\geq 2$. Dada uma tarefa $x\in I'$, para toda data $h\in \bigcup_{y\in I'\setminus\{x\}} H_y$ e todo $\Delta\in\Big\{2-P_x-$*

$\min\limits_{y\in I'\setminus\{x\}} S_{xy}, \ldots, \max\limits_{y\in I'\setminus\{x\}} \big(P_y + S_{yx}\big)\Big\}$, *tem-se:*

$$\underbrace{\sum_{k=\lfloor h-P_x-S_x^{min}+1\rfloor_x}^{\lceil h+\Delta-1\rceil_x} l_{xk}}_{\varepsilon_x} + \sum_{y\in I^*} \underbrace{\sum_{k=\lfloor h-P_y-S_y^{min}+\Delta\rfloor_y}^{\lceil h\rceil_y} l_{yk}}_{\epsilon_y} \;\leq\; 1, \tag{4.67}$$

em que

- $I^* = \{y \in I' \setminus \{x\} \mid P_y + S_{yx} \geq \Delta\}$,
- $S_x^{min} = \min\limits_{y\in I^*} S_{xy}$ *e*
- $S_y^{min} = \min\left(S_{yx},\ \Delta - 1 + \min\limits_{z\in I^*\setminus\{y\}} S_{yz}\right)$ *para toda tarefa* $y \in I^*$ $\big($*se* $I^* = \{y\}$, *então* $S_y^{min} = S_{yx}\big)$.

Prova: Dado que a tarefa x deve ser executada uma única vez, tem-se $\varepsilon_x \leq 1$. Além disso, segue das restrições (4.63) que $\sum_{y\in I^*} \epsilon_y \leq 1$. Suponha que exista uma programação π de I tal que $\varepsilon_x = 1$ e $\sum_{y\in I^*} \epsilon_y = 1$ para uma dada data h' e um dado Δ'. Logo, existe uma tarefa $y' \in I^*$ tal que $\epsilon_{y'} = 1$. Consequentemente, $h' - P_x - S_{xy'} + 1 \leq s_x^\pi \leq h' + \Delta' - 1$ e $h' - P_{y'} - S_{y'x} + \Delta' \leq s_{y'}^\pi \leq h'$, isto é, a máquina executa as tarefas x e y' simultaneamente na programação π. Isso contradiz o fato de π ser uma programação factível de I. □

Os conjuntos de restrições (4.58), (4.63) e (4.65) estão contidos na Família 1. As restrições (4.58) são obtidas ao se considerar todos os subconjuntos $I' \subset I$ tais que $|I'| = 2$ e, para todo $x \in I'$, tomar $\Delta = 2 - P_x - S_{xy}$, em que y é tal que $\{x\} \cup \{y\} = I'$. Por outro lado, se for considerado apenas $\Delta = 1$ na Família 1, obtêm-se as restrições (4.63). Finalmente, as restrições (4.65) são obtidas da Família 1 assim como as restrições (4.65), exceto pelo valor de Δ, que, para todo $x \in I'$, deve ser fixado em $\Delta = P_y + S_{yx} - P_x - S_{xy} + 1$, para cada $x \in I'$, em que y é tal que $\{x\} \cup \{y\} = I'$.

O lema a seguir provê um novo conjunto de restrições válidas.

Lema 4.1 *Para todo subconjunto* $I' \subseteq I$, *tem-se:*

$$\sum_{x\in I'} \underbrace{\left(\sum_{k=\lfloor h\rfloor_x}^{\lceil h+\min\limits_{y\in I'\setminus\{x\}}(P_y+S_{yx})-1\rceil_x} l_{xk}\right)}_{\epsilon_x} \;\leq\; 1, \quad \forall\, h \in \bigcup_{x\in I'} H_x. \tag{4.68}$$

Prova: Posto que cada tarefa deve ser executada uma única vez, tem-se $\epsilon_x \leq 1$, $\forall x \in I'$ e $\forall h \in \bigcup_{x\in I'} H_x$. Suponha que exista uma programação π de I tal que $\sum_{x\in I'} \epsilon_x > 1$ para uma dada data h'. Logo, existem duas tarefas $x_1, x_2 \in I'$ tais que $\epsilon_{x_1} = \epsilon_{x_2} = 1$. Consequentemente, $h' \leq s_{x_1}^\pi \leq h' + P_{x_2} + S_{x_2,x_1} - 1$ e $h' \leq s_{x_2}^\pi \leq h' + P_{x_1} + S_{x_1,x_2} - 1$, ou seja, a máquina executa as tarefas x_1 e x_2 simultaneamente na programação π. Isso contradiz o fato de π ser uma programação factível de I. □

Observe que o conjunto de restrições (4.65) está contido no conjunto (4.68), sendo obtido ao se considerar somente os subconjuntos $I' \subset I$ tais que $|I'| = 2$.

A Proposição 4.3 fornece outra família de restrições válidas, denominada "Família 2". Essa família contém o conjunto de restrições (4.68).

Proposição 4.3 (Família 2) *Seja* $I' \subseteq I$ *um subconjunto de tarefas tal que* $|I'| \geq 2$. *Dada uma tarefa* $x \in I'$, *para toda data* $h \in \bigcup_{y \in I' \setminus \{x\}} H_y$ *e todo* $\Delta \in \left\{ 2 - \min_{y \in I^*} \left(P_y + S_{yx}\right), \ldots, P_x + \max_{y \in I' \setminus \{x\}} S_{xy} \right\}$, *tem-se:*

$$\underbrace{\sum_{k=\lfloor h-\Delta+1 \rfloor_x}^{\lceil h+PS_x^{min}-1 \rceil_x} l_{xk}}_{\varepsilon_x} + \sum_{y \in I^*} \underbrace{\sum_{k=\lfloor h \rfloor_y}^{\lceil h+PS_y^{min}-\Delta \rceil_y} l_{yk}}_{\epsilon_y} \leq 1, \tag{4.69}$$

em que:

- $I^* = \{y \in I' \setminus \{x\} \mid P_x + S_{xy} \geq \Delta\}$,
- $PS_x^{min} = \min_{y \in I^*} \left(P_y + S_{yx}\right)$ *e*
- *Se* $I^* = \{y\}$, *então* $PS_y^{min} = P_x + S_{xy}$. *Por outro lado, se* $|I^*| \geq 2$, *então* $PS_y^{min} = \min\left(P_x + S_{xy},\ \Delta - 1 + \min_{z \in I^* \setminus \{y\}} \left(P_z + S_{zy}\right)\right)$, *para todo* $y \in I^*$.

Prova: Uma vez que a tarefa x deve ser executada uma única vez, tem-se $\varepsilon_x \leq 1$. Ademais, segue das restrições (4.68) que $\sum_{y \in I^*} \epsilon_y \leq 1$. Suponha que exista uma programação π de I tal que $\varepsilon_x = 1$ e $\sum_{y \in I^*} \epsilon_y = 1$ para uma dada data h' e um dado Δ'. Logo, existe uma tarefa $y' \in I^*$ tal que $\epsilon_{y'} = 1$. Consequentemente, $h' - \Delta' + 1 \leq s_x^\pi \leq h' + P_{y'} + S_{y'x} - 1$ e $h' \leq s_{y'}^\pi \leq h' + P_x + S_{xy'} - \Delta'$, ou seja, a máquina executa as tarefas x e y' simultaneamente na programação π. Isso contradiz o fato de π ser uma programação factível de I. □

As restrições (4.68) são obtidas da Família 2 ao se considerar $\Delta = 1$. Os conjuntos de restrições (4.58) e (4.64) também estão contidos na Família 2. As restrições (4.58) são obtidas ao se considerar todos os subconjuntos $I' \subset I$ tais que $|I'| = 2$ e, para todo $x \in I'$, tomar $\Delta = 2 - P_y - S_{yx}$, em que y é tal que $\{x\} \cup \{y\} = I'$. As restrições (4.65) são obtidas da Família 2 analogamente às restrições (4.64), exceto pelo valor de Δ, que, para todo $x \in I'$, deve ser fixado em $\Delta = P_x + S_{xy} - P_y - S_{yx} + 1$, para cada $x \in I'$, em que y é tal que $\{x\} \cup \{y\} = I'$.

As Proposições 4.4, 4.5 e 4.6 fornecem mais três famílias de restrições válidas, as quais serão denominadas "Família 3", "Família 4" e "Família 5", respectivamente.

Proposição 4.4 (Família 3) *Para todo subconjunto de tarefas* $I' \subseteq I$ *tal que* $|I'| \geq 2$, *tem-se:*

$$\sum_{x \in I'} \underbrace{\sum_{k=s_x^{LB}}^{\left\lceil \min_{y \in I' \setminus \{x\}} (s_y^{LB} + P_y + S_{yx}) - 1 \right\rceil_x} l_{xk}}_{\epsilon_x} \leq 1. \tag{4.70}$$

Prova De acordo com o que já foi discutido, $\epsilon_x \leq 1$, $\forall x \in I'$. Suponha que exista uma programação π de I tal que $\sum_{x \in I'} \epsilon_x > 1$. Logo, existem duas tarefas $x_1, x_2 \in I'$ tais que $\epsilon_{x_1} = \epsilon_{x_2} = 1$. Consequentemente, $s_{x_1}^{LB} \leq s_{x_1}^\pi \leq s_{x_2}^{LB} + P_{x_2} + S_{x_2,x_1} - 1$ e $s_{x_2}^{LB} \leq s_{x_2}^\pi \leq s_{x_1}^{LB} + P_{x_1} + S_{x_1,x_2} - 1$, ou seja, a máquina executa as tarefas x_1 e x_2 simultaneamente na programação π. Isso contradiz o fato de π ser uma programação factível de I. □

Proposição 4.5 (Família 4) *Dado um subconjunto de tarefas* $I' \subseteq I$ *tal que* $|I'| \geq 2$, *se* $s^{LB}_{I'} = \min_{x \in I'} s^{LB}_x$ *e* $TPT^{I'}_{min}$ *representa o menor tempo total necessário para executar todas as tarefas em* I', *então*

$$\underbrace{\sum_{x \in I'} \sum_{h=\left\lfloor s^{LB}_{I'\setminus\{x\}} + TPT^{I'\setminus\{x\}}_{min} + \min_{y \in I\setminus\{x\}} S_{yx} \right\rfloor_x}^{s^{UB}_x} l_{xh}}_{\epsilon_{I'}} \quad \geq \quad 1. \tag{4.71}$$

Prova: Suponha que exista uma programação viável π de I' tal que $\epsilon_{I'} = 0$. Seja $x' \in I'$ a última tarefa de I' executada em π. Logo, $s^{\pi}_{x'} - \min_{y \in I\setminus\{x'\}} S_{yx'} \leq s^{LB}_{I'\setminus\{x\}} + TPT^{I'\setminus\{x'\}}_{min} - 1$ e existe uma programação das tarefas em $I' \setminus \{x'\}$ cujo tempo tempo total de execução é menor que $TPT^{I'\setminus\{x'\}}_{min}$. Isso contradiz o fato de $TPT^{I'\setminus\{x'\}}_{min}$ ser o menor tempo total necessário para executar as tarefas em $I' \setminus \{x'\}$. □

Proposição 4.6 (Família 5) *Se* x *e* y *são duas tarefas distintas de* I, *então*

$$\underbrace{\sum_{k=h+P_x+S_{xy}}^{s^{UB}_y} l_{yk}}_{\epsilon_y} \quad \geq \quad l_{xh}, \qquad \forall h \in \{s^{LB}_y, s^{LB}_y + 1, \ldots, s^{LB}_y + P_y + S_{yx} - 1\} \cap H_x. \tag{4.72}$$

Prova: Suponha que exista uma programação viável π de I tal que $\epsilon_y < l_{xh'}$ para algum $h' \in \{s^{LB}_y, s^{LB}_y + 1, \ldots, s^{LB}_y + P_y + S_{yx} - 1\} \cap H_x$. Logo, $l_{xh'} = 1$ e $\epsilon_y = 0$. Dado que $h' = s^{\pi}_x$, tem-se $s^{\pi}_x \leq s^{LB}_y + P_y + S_{yx} - 1$ e $s^{\pi}_y \leq s^{\pi}_x + P_x + S_{xy} - 1$, isto é, a máquina executa as tarefas x e y simultaneamente na programação π. Isso contradiz o fato de π ser uma programação viável de I. □

4.7 Algoritmos de separação para as novas famílias de restrições baseadas em variáveis indexadas no tempo

Com exceção da Família 5, todas as famílias de restrições propostas na Seção 4.6 possuem um número elevado de restrições (da ordem de 2^n ou maior). Tal fato inviabiliza a inclusão completa dessas famílias de restrições nas formulações indexadas no tempo para o SMSPETP-SDS. No entanto, tais famílias podem ser utilizadas em algoritmos de planos-de-corte (Wolsey, 1998) para o problema. Em resumo, algoritmos de planos-de-corte são procedimentos que partem da solução de um problema de programação matemática sem restrições (ou com apenas algumas restrições), ao qual, iterativamente, são acrescidas novas restrições válidas para o problema a ser resolvido, até que um critério de parada seja satisfeito.

Seja PPM o problema de programação matemática baseado em variáveis indexadas no tempo que é atualizado iterativamente em um dado algoritmo de planos-de-corte. Considere que l^* representa uma solução ótima do PPM corrente. Observe que l^* consiste em uma matriz de valores atribuídos às variáveis l_{xh}, $\forall x \in I$ e $\forall h \in H_x$. Devido ao grande número de restrições nas Famílias 1–4, o simples fato de verificar uma a uma quais restrições são violadas por l^* é, ainda, um processo impraticável. O problema de encontrar, em um conjunto de restrições, aquelas que são violadas por l^* é chamado de "problema de separação".

O problema de separação associado à Família 5 é resolvido de forma exata, verificando todas as restrições, uma a uma. Por outro lado, os problemas de separação das Famílias 1–4

são resolvidos de forma heurística. Além disso, é considerado uma violação mínima, a qual é determinada por um parâmetro real $\delta > 0$. Uma restrição do tipo $M \times l \leq b$ é violada por l^* por, no mínimo, δ se $M \times l^* \geq b + \delta$. Analogamente, uma restrição do tipo $M \times l \geq b$ é violada por l^* por δ se $M \times l^* \leq b - \delta$.

Os algoritmos propostos para resolver os problemas de separação associados às famílias de restrições propostas na Seção 4.6 são descritos nas subseções a seguir. Dada uma solução l^* para o PPM corrente, considere $h_x^{min} = \min\{h \in H_x : l^*_{xh} > 0\}$ e $h_x^{max} = \max\{h \in H_x : l^*_{xh} > 0\}$, $\forall x \in I$. Ademais, para toda tarefa $x \in I$, considere $I_x = \left\{y \in I \setminus \{x\} : h_y^{max} + P_y + S_{yx} > h_x^{min} \text{ ou } h_x^{max} + P_x + S_{xy} > h_y^{min}\right\}$.

4.7.1 Heurística de separação para a Família 1

O algoritmo heurístico proposto para o problema de separação da Família 1 é descrito no Algoritmo 4.1. Nele, Ω_1 representa o conjunto de restrições encontradas por esse algoritmo, isto é, Ω_1 é um subconjunto da Família 1 composto de restrições que são violadas por l^*. $lhs_{x,h,I',\Delta}(l^*)$ representa o valor numérico da expressão $\varepsilon_x + \sum_{y \in I^*} \epsilon_y$ da Proposição 4.2 aplicada a l^*, para os respectivos x, h, I' e Δ. δ representa a violação mínima considerada.

De acordo com a Proposição 4.2, cada restrição da Família 1 é determinada por um conjunto $I' \subseteq I$, uma tarefa $x \in I'$, uma data h e um valor de Δ.

Como pode ser visto no Algoritmo 4.1, para toda tarefa $x \in I$ e toda data $h \in \{h_x^{max}, h_x^{max} - 1, \cdots, h_x^{min}\}$, a heurística de separação proposta para a Família 1 tenta construir um conjunto I' de modo que a restrição definida por x, h, I' e Δ seja violada por l^*, para algum valor de Δ. A construção de I' é inicializada com $I' = \{x\}$ e, a seguir, verifica-se o ganho de se inserir cada uma das tarefas de I_x em I'. O ganho de inserir uma tarefa y em I' é calculado por meio da função $lhs_{x,h,I',\Delta}(l^*)$ aplicada para todos os valores de $\Delta \in \{P_y + S_{yx}, P_y + S_{yx} - 1, \cdots, 2 - P_x - \min_{z \in I \setminus \{x\}} S_{xy}\}$. Se $lhs_{x,h,I',\Delta}(l^*) \geq 1 + \delta$ para algum Δ ao verificar a inserção da tarefa y em I', então a restrição associada aos atuais x, h, I' e Δ é violada por l^* e pertencerá ao conjunto de restrições retornado pelo algoritmo (Ω_1). Nesse caso, a rotina é interrompida com sucesso para a tarefa x e data h correntes. Caso contrário, isto é, se $lhs_{x,h,I',\Delta}(l^*) < 1 + \delta$, se inserida qualquer tarefa de $I_x \setminus I'$ em I', então a tarefa $y*$ associada ao maior ganho é inserida definitivamente em I' e o procedimento de análise de ganho é repetido para as tarefas de $I_x \setminus I'$ no novo conjunto I'. Além disso, se o ganho associado a y^* for igual a zero, isto é, se a inserção de y^* em I' não aumenta o valor de $lhs_{x,h,I',\Delta}(l^*)$, então a rotina é interrompida sem sucesso para a tarefa x e a data h correntes.

A ordem de investigação das tarefas $x \in I$ é sempre dada pela ordem crescente de h_x^{min}. Ademais, o número máximo de restrições retornadas pela heurística de separação da Família 1 é dada por $\sum_{x \in I}(h_x^{max} - h_x^{min} + 1)$.

4.7.2 Heurística de separação para a Família 2

O algoritmo heurístico proposto para o problema de separação da Família 2 é detalhado no Algoritmo 4.1. Nele, Ω_2 representa o conjunto de restrições encontradas por esse algoritmo, isto é, Ω_2 é um subconjunto da Família 2 composto de restrições que são violadas por l^*. $lhs_{x,h,I',\Delta}(l^*)$ representa o valor numérico da expressão $\varepsilon_x + \sum_{y \in I^*} \epsilon_y$ da Proposição 4.3 aplicada a l^*, para os respectivos x, h, I' e Δ. δ representa a violação mínima considerada.

Assim como as restrições da Família 1, cada restrição da Família 2 é determinada por um conjunto $I' \subseteq I$, uma tarefa $x \in I'$, uma data h e um valor de Δ. Logo, o algoritmo heurístico proposto para o problema de separação da Família 2, detalhado no Algoritmo 4.2, é análogo à heurística de separação proposta para a Família 1. As únicas diferenças encontram-se no intervalo de variação de Δ e na função $lhs_{x,h,I',\Delta}(l^*)$, que, neste caso, baseiam-se na

Algoritmo 4.1 Heurística de separação para a Família 1.

$\Omega_1 \leftarrow \emptyset$;
para $x \in I$ **faça**
  para $h = h_x^{max}, h_x^{max} - 1, \cdots, h_x^{min}$ **faça**
    $I' \leftarrow \{x\}$;
    $lhs_0 \leftarrow -\infty$;
    Atualizou $\leftarrow$ `NÃO`;
    enquanto $I' \neq I_x \cup \{x\}$ **faça**
      $y^* \leftarrow -1$;
      $lhs^* \leftarrow -\infty$;
      para $y \in I_x \setminus I'$ **faça**
        $I' \leftarrow I' \cup \{y\}$;
        para $\Delta = P_y + S_{yx}, P_y + S_{yx} - 1, \cdots, 2 - P_x - \min_{z \in I \setminus \{x\}} S_{xz}$ **faça**
          se $lhs_{x,h,I',\Delta}(l^*) \geq 1 + \delta$ **então**
            $\Omega_1 = \Omega_1 \cup \{lhs_{x,h,I',\Delta}(l^*) \leq 1\}$;
            Atualizou $\leftarrow$ `SIM`;
            Sai do laço atual;
          fim
          se $lhs_{x,h,I',\Delta}(l^*) > lhs^*$ **então**
            $lhs^* \leftarrow lhs_{x,h,I',\Delta}(l^*)$;
            $y^* \leftarrow y$;
          fim
        fim
        se Atualizou = `SIM` **então**
          Sai do laço atual;
        senão
          $I' \leftarrow I' \setminus \{y\}$;
        fim
      fim
      se Atualizou = `SIM` **então**
        Sai do laço atual;
      fim
      se $lhs^* > lhs_0$ **então**
        $lhs_0 \leftarrow lhs^*$;
      senão
        Sai do laço atual;
      fim
      $I' \leftarrow I' \cup \{y^*\}$;
    fim
  fim
fim
Retorne Ω_1;

Proposição 4.3.

Igualmente à heurística de separação da Família 1, a ordem de investigação das tarefas $x \in I$ é sempre dada pela ordem crescente de h_x^{min} e o número máximo de restrições retornadas pela heurística de separação da Família 2 é igual a $\sum_{x \in I}(h_x^{max} - h_x^{min} + 1)$.

Algoritmo 4.2 Heurística de separação para a Família 2.

$\Omega_2 \leftarrow \emptyset$;
para $x \in I$ **faça**
  para $h = h_x^{max}, h_x^{max} - 1, \cdots, h_x^{min}$ **faça**
    $I' \leftarrow \{x\}$;
    $lhs_0 \leftarrow -\infty$;
    Atualizou $\leftarrow$ `NÃO`;
    enquanto $I' \neq I_x \cup \{x\}$ **faça**
      $y^* \leftarrow -1$;
      $lhs^* \leftarrow -\infty$;
      para $y \in I_x \setminus I'$ **faça**
        $I' \leftarrow I' \cup \{y\}$;
        para $\Delta = P_x + S_{xy}, P_x + S_{xy} - 1, \cdots, 2 - \min_{z \in I \setminus \{x\}}(P_z + S_{zx})$ **faça**
          se $lhs_{x,h,I',\Delta}(l^*) \geq 1 + \delta$ **então**
            $\Omega_2 = \Omega_2 \cup \{lhs_{x,h,I',\Delta}(l) \leq 1\}$;
            Atualizou $\leftarrow$ `SIM`;
            Sai do laço atual;
          fim
          se $lhs_{x,h,I',\Delta}(l^*) > lhs^*$ **então**
            $lhs^* \leftarrow lhs_{x,h,I',\Delta}(l^*)$;
            $y^* \leftarrow y$;
          fim
        fim
        se Atualizou = `SIM` **então**
          Sai do laço atual;
        senão
          $I' \leftarrow I' \setminus \{y\}$;
        fim
      fim
      se Atualizou = `SIM` **então**
        Sai do laço atual;
      fim
      se $lhs^* > lhs_0$ **então**
        $lhs_0 \leftarrow lhs^*$;
      senão
        Sai do laço atual;
      fim
      $I' \leftarrow I' \cup \{y^*\}$;
    fim
  fim
fim
Retorne Ω_2;

4.7.3 Heurística de separação para a Família 3

O algoritmo heurístico proposto para o problema de separação da Família 3 é detalhado no Algoritmo 4.3. Nele, Ω_3 representa o conjunto de restrições encontradas por esse algoritmo, isto é, Ω_3 é um subconjunto da Família 3 composto de restrições que são violadas por l^*. $lhs_{I'}(l^*)$ representa o valor numérico da expressão $\sum_{x \in I'} \epsilon_x$ da Proposição 4.4 aplicada a l^*, para o respectivo subconjunto $I' \subseteq I$. Novamente, δ representa a violação mínima considerada.

De acordo com a Proposição 4.4, cada restrição da Família 3 é determinada por um con-

Algoritmo 4.3 Heurística de separação para a Família 3.

$\Omega_3 \leftarrow \emptyset$;
para $x \in I$ **faça**
  $I' \leftarrow \{x\}$;
  $lhs_0 \leftarrow -\infty$;
  Atualizou $\leftarrow$ `NÃO`;
  enquanto $I' \neq I_x \cup \{x\}$ **faça**
    $y^* \leftarrow -1$;
    $lhs^* \leftarrow -\infty$;
    para $y \in I_x \setminus I'$ **faça**
      $I' \leftarrow I' \cup \{y\}$;
      se $lhs_{I'}(l^*) \geq 1 + \delta$ **então**
        $\Omega_3 = \Omega_3 \cup \{lhs_{I'}(l^*) \leq 1\}$;
        Atualizou $\leftarrow$ `SIM`;
        Sai do laço atual;
      fim
      se $lhs_{I'}(l^*) > lhs^*$ **então**
        $lhs^* \leftarrow lhs_{I'}(l^*)$;
        $y^* \leftarrow y$;
      fim
      $I' \leftarrow I' \setminus \{y\}$;
    fim
    se Atualizou = `SIM` **então**
      Sai do laço atual;
    fim
    se $lhs^* > lhs_0$ **então**
      $lhs_0 \leftarrow lhs^*$;
    senão
      Sai do laço atual;
    fim
    $I' \leftarrow I' \cup \{y^*\}$;
  fim
fim
Retorne Ω_3;

junto $I' \subseteq I$.

Conforme o Algoritmo 4.3, para toda tarefa $x \in I$, a heurística de separação proposta para a Família 3 tenta construir um conjunto I' de modo que a restrição definida por I' seja violada por l^*. A construção de I' é inicializada com $I' = \{x\}$ e, a seguir, verifica-se o ganho de se inserir cada uma das tarefas de I_x em I'. O ganho de inserir uma tarefa y em I' é calculado por meio da função $lhs_{I'}(l^*)$. Se $lhs_{I'}(l^*) \geq 1 + \delta$ ao verificar a inserção da tarefa y em I', então a restrição associada ao atual I' é violada por l^* e pertencerá ao conjunto de restrições retornado pelo algoritmo (Ω_3). Nesse caso, a rotina é interrompida com sucesso para a tarefa x corrente. Caso contrário, isto é, se $lhs_{I'}(l^*) < 1+\delta$, se inserida qualquer tarefa de $I_x \setminus I'$ em I', então a tarefa $y*$ associada ao maior ganho é inserida definitivamente em I' e a procedimento de análise de ganho é repetido para as tarefas de $I_x \setminus I'$ no novo conjunto I'. Além disso, se o ganho associado a y^* for igual a zero, isto é, se a inserção de y^* em I' não aumenta o valor de $lhs_{I'}(l^*)$, então a rotina é interrompida sem sucesso para a tarefa x corrente.

Assim como nas heurísticas de separação para as Famílias 1 e 2, a ordem de investigação das tarefas $x \in I$ é sempre dada pela ordem crescente de h_x^{min}. Já o número máximo de restrições retornadas pela heurística de separação da Família 3 é igual a n.

4.7.4 Heurística de separação para a Família 4

Antes de apresentar a heurística de separação proposta para a Família 4, observa-se que a restrição (4.71) da Proposição 4.5 é equivalente à restrição (4.73), para todo subconjunto $I' \subseteq I$. Tal equivalência é obtida por meio das relações (4.52).

$$\underbrace{\sum_{x \in I'} \sum_{h=s_x^{LB}}^{\left\lceil s_{I'\setminus\{x\}}^{LB} + TPT_{min}^{I'\setminus\{x\}} + \min\limits_{y \in I\setminus\{x\}} S_{yx} - 1 \right\rceil_x} l_{xh}}_{\epsilon'_{I'}} - |I'| \;\leq\; -1. \tag{4.73}$$

O algoritmo heurístico proposto para o problema de separação da Família 4 é descrito no Algoritmo 4.4. Nele, Ω_4 representa o conjunto de restrições encontradas por esse algoritmo, isto é, Ω_4 é um subconjunto da Família 4 composto de restrições que são violadas por l^*. $lhs_{I'}(l^*)$ representa o valor numérico da expressão $\epsilon'_{I'} - |I'|$ da restrição (4.73) aplicada a l^*, para o respectivo subconjunto $I' \subseteq I$. Mais uma vez, δ representa a violação mínima considerada.

Algoritmo 4.4 Heurística de separação para a Família 4.

```
Ω4 ← ∅;
para x ∈ I faça
    I' ← {x};
    lhs0 ← −∞;
    Atualizou ← NÃO;
    enquanto I' ≠ Ix ∪ {x} faça
        y* ← −1;
        lhs* ← −∞;
        para y ∈ Ix \ I' faça
            I' ← I' ∪ {y};
            se lhs_I'(l*) ≥ −1 + δ então
                Ω4 = Ω4 ∪ {lhs_I'(l*) ≤ −1};
                Atualizou ← SIM;
                Sai do laço atual;
            fim
            se lhs_I'(l*) > lhs* então
                lhs* ← lhs_I'(l*);
                y* ← y;
            fim
            I' ← I' \ {y};
        fim
        se Atualizou = SIM então
            Sai do laço atual;
        fim
        se lhs* > lhs0 então
            lhs0 ← lhs*;
        senão
            Sai do laço atual;
        fim
        I' ← I' ∪ {y*};
    fim
fim
Retorne Ω4;
```

Tal como nas restrições da Família 3, cada restrição da Família 4 é determinada por um conjunto $I' \subseteq I$. Desse modo, o algoritmo heurístico proposto para o problema de separação da Família 4, detalhado no Algoritmo 4.4, é análogo à heurística de separação proposta para a Família 3. A única diferença encontra-se na função $lhs_{I'}(l^*)$, que, neste caso, baseia-se na restrição (4.73). Além disso, em vez do valor exato de $TPT^{I'}_{min}$, é utilizado uma cota inferior para esse valor. A cota utilizada no lugar de $TPT^{I'}_{min}$ é fornecida pelo Corolário 4.1. Tal corolário é uma consequência direta da Proposição 2.2.

Corolário 4.1 *Para todo subconjunto $I' \subseteq I$, o menor tempo total necessário para executar todas as tarefas de I', $TPT^{I'}_{min}$, é tal que*

$$TPT^{I'}_{min} \geq \sum_{x \in I'} P_x + \max\left(\sum_{x \in I'} \min_{y \in I' \setminus \{x\}} S_{yx} - \max_{x \in I'} \min_{y \in I' \setminus \{x\}} S_{yx}, \sum_{x \in I'} \min_{y \in I' \setminus \{x\}} S_{xy} - \max_{x \in I'} \min_{y \in I' \setminus \{x\}} S_{xy} \right).$$

Da mesma forma que nas heurísticas de separação das Famílias 1–3, a ordem de investigação das tarefas $x \in I$ é sempre dada pela ordem crescente de h^{min}_x. Além disso, o número máximo de restrições retornadas pela heurística de separação da Família 4 é igual a n.

4.7.5 Algoritmo de separação para a Família 5

Como já mencionado anteriormente, diferentemente dos demais problemas de separação, o problema de separação da Família 5 é resolvido de forma exata. Dada uma solução l^* para o PPM corrente, verifica-se uma a uma quais as restrições da Família 5 que são violadas por l^*.

O algoritmo proposto para o problema de separação da Família 5 é detalhado no Algoritmo 4.5. Nele, Ω_5 representa o conjunto de restrições encontradas por esse algoritmo, ou seja, Ω_5 é um subconjunto da Família 5 composto de restrições que são violadas por l^*. Novamente, δ representa a violação mínima considerada.

Algoritmo 4.5 Algoritmo de separação para a Família 5.

$\Omega_5 \leftarrow \emptyset$;
para $x \in I$ **faça**
  para $y \in I \setminus \{x\}$ **faça**
    para $h = \max(s^{LB}_x, s^{LB}_y), \max(s^{LB}_x, s^{LB}_y) + 1, \cdots, \min(s^{UB}_x, s^{LB}_y + P_y + S_{yx} - 1)$ **faça**
      se $\sum_{k=h+P_x+S_{xy}}^{s^{UB}_y} l^*_{yk} \leq l^*_{xh} - \delta$ **então**
        $\Omega_5 = \Omega_5 \cup \left\{ \sum_{k=h+P_x+S_{xy}}^{s^{UB}_y} l_{yk} \geq l_{xh} \right\}$;
      fim
    fim
  fim
fim
Retorne Ω_5;

4.8 Resultados computacionais

Nesta seção são apresentados e discutidos os resultados computacionais obtidos com as formulações de programação matemática para o SMSPETP-SDS apresentadas nas Seções 4.1–4.4, bem como com os algoritmos de separação propostos na Seção 4.7.

As formulações matemáticas foram implementadas e resolvidas por meio da ferramenta C++ Concert Technology e do otimizador IBM ILOG CPLEX Optimization Studio 12.6.2.

Os experimentos com as heurísticas de separação também foram implementados na linguagem C++, sendo utilizado o compilador g++, versão 4.8.5, para a sua execução. Os experimentos foram realizados em um computador Intel® Xeon(R) CPU E5620 @ 2.40GHz × 16, com 48 GB de RAM e sistema operacional CentOS Linux 7. O CPLEX foi configurado para utilizar somente uma *thread* e os demais parâmetros não foram alterados, exceto quando evidenciado. Além disso, os algoritmos não foram otimizados para multiprocessamento.

Os experimentos computacionais foram realizados com os problemas-teste descritos na Seção 2.2. Foram utilizados os conjuntos de problemas com até 20 tarefas, sendo 16 problemas-teste em cada conjunto de problemas com a mesma quantidade de tarefas. Para cada tarefa $x \in I$, os limites s_x^{UB} e s_x^{LB} utilizados para determinar os parâmetros de entrada de cada formulação matemática são aqueles fornecidos pelo Teorema 2.2.

Dado um problema-teste, o *gap* de um dado limite inferior LB para esse problema em relação a uma dada solução π^* desse mesmo problema é dado pela Equação (4.74):

$$gap = \frac{f(\pi^*) - LB}{f(\pi^*)} \times 100\%, \tag{4.74}$$

em que $f(\pi^*)$ representa o valor da função objetivo do problema aplicada a π^*. Logo, quanto mais próximo de zero estiver o *gap* de LB, mais próximo esse limite inferior estará do valor da função objetivo da solução π^*. Em outras palavras, quanto menor for o valor do *gap*, melhor é o limite inferior LB.

As melhores soluções encontradas nos experimentos computacionais apresentados na Seção 3.4 foram utilizadas como referências para os cálculos dos respectivos *gap*'s.

O restante desta seção está organizado da seguinte forma. Na Seção 4.8.1 são apresentados e analisados os resultados obtidos com as relaxações lineares das formulações matemáticas apresentadas nas Seções 4.1–4.4, enquanto a Seção 4.8.2 apresenta e discute os resultados obtidos com os algoritmos de separação propostos na Seção 4.7.

4.8.1 Resultados obtidos com as relaxações lineares das formulações apresentadas nas Seções 4.1–4.4

Na primeira bateria de experimentos, utilizou-se o CPLEX para resolver as relaxações lineares das formulações VP, VP', VPI, VPIA, VPS, IMR, IMN, BR, BN, B1 e B2 (ver Seções 4.1–4.4) aplicadas aos problemas-testes. O tempo para o CPLEX resolver a relaxação linear de cada problema-teste foi limitado em uma hora para todas as formulações.

As soluções das relaxações lineares das formulações matemáticas para o SMSPETP-SDS aplicadas aos problemas-teste foram utilizadas como limites inferiores (LB) nos cálculos dos respectivos *gap*'s. A Tabela 4.2 apresenta as médias dos *gap*'s obtidos (em porcentagem) com as respectivas relaxações lineares aplicadas aos problemas de cada conjunto de problemas-teste com o mesmo número de tarefas. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste com a mesma quantidade de tarefas.

De acordo com a Tabela 4.2, as médias dos *gap*'s das soluções obtidos pelo CPLEX com as relaxações lineares das formulações VP, VPI, VPS, IMR e IMN foram sempre próximas. Com essas formulações, os *gap*'s médios das soluções obtidas foram sempre superiores a 88%. A formulação VP', obtida da formulação VP por meio da inclusão de uma nova família de restrições, foi a que obteve soluções relaxadas de menores *gap*'s médios para os problemas com 12, 16, 17, 19 e 20 tarefas. Para os demais problemas, a formulação B2 foi a que obteve soluções relaxadas de menores *gap*'s médios. A relaxação linear da formulação VP' encontrou soluções de *gap*'s médios sempre significativamente menores que os respectivos *gap*'s médios das soluções obtidas com a relaxação linear da formulação VP. Para os problemas-teste com 9 tarefas, por exemplo, a relaxação linear formulação VP obteve soluções de *gap* médio igual a 94,86% ao

Tabela 4.2: *gap*'s médios obtidos com as relaxações lineares das formulações apresentadas nas Seções 4.1–4.4 (em porcentagem).

n	VP	VP'	VPI	VPIA	VPS	IMR	IMN	BR	BN	B1	B2
06	93,47	73,66	93,47	89,50	92,57	92,47	91,16	85,68	73,79	37,85	29,64
07	92,62	63,03	92,62	89,33	92,34	91,86	90,62	87,49	78,18	49,03	45,65
08	90,66	66,69	90,66	87,76	90,02	89,74	88,67	86,25	78,61	55,46	50,93
09	94,86	59,19	94,86	92,57	94,84	94,01	93,14	90,41	82,61	56,69	54,15
10	94,84	66,08	94,84	93,29	94,84	94,22	93,64	91,06	84,55	58,47	55,30
11	96,11	70,01	96,11	94,92	96,11	95,75	95,32	91,86	85,60	64,28	60,56
12	94,32	62,11	94,32	92,47	94,32	93,62	92,83	91,18	85,98	68,74	65,43
13	93,30	64,36	93,30	91,32	93,30	92,44	91,47	89,48	83,62	63,79	59,85
14	94,48	68,42	94,48	93,39	94,48	93,94	93,42	91,31	85,62	64,79	61,69
15	95,86	68,09	95,86	94,73	95,86	95,11	94,35	92,43	87,53	70,10	67,32
16	94,12	64,47	94,12	92,69	94,12	93,58	93,06	91,13	86,79	71,55	69,25
17	93,90	63,22	93,90	92,69	93,90	93,38	92,84	91,35	87,53	73,08	70,25
18	94,94	73,11	94,94	94,13	94,94	94,70	94,36	92,40	87,24	69,07	66,81
19	95,29	64,85	95,29	94,53	95,29	94,94	94,78	92,80	88,55	71,70	69,18
20	95,95	68,48	95,95	94,96	95,95	95,58	95,19	93,56	89,64	74,54	72,17

passo que a formulação VP' obteve soluções de *gap* médio igual a 59,19%. A relaxação linear da formulação VPIA também encontrou soluções de *gap*'s médios sempre inferiores que os respectivos *gap*'s médios das soluções encontradas com a relaxação linear da formulação VPI. No entanto, as médias dos *gap*'s das soluções obtidas com a relaxação linear da formulação VPIA foram sempre superiores a 87%. Comportamento semelhante ocorreu com a relaxação da formulação IMN, que obteve soluções de *gap*'s médios ligeiramente menores que os respectivos *gap*'s médios das soluções obtidas com a relaxação linear da formulação IMR. As formulações binárias BR e BN, obtidas respectivamente das formulações IMR e IMN por meio da eliminação das variáveis reais, encontraram soluções relaxadas de *gap*'s médios inferiores aos respectivos *gap*'s médios das soluções relaxadas encontradas pelas formulações IMR e IMN. Finalmente, a relaxação linear da formulação binária B1 encontrou soluções de *gap*'s médios maiores que os respectivos *gap*'s médios das soluções encontradas com a relaxação linear da formulação B2.

Embora não seja mostrado na Tabela 4.2, as relaxações lineares das formulações binárias B1 e B2 encontraram soluções de *gap*'s iguais a 0% para dois problemas-teste com 6 tarefas e um problema-teste com 9 tarefas, isto é, as soluções das relaxações lineares dessas formulações aplicadas a tais problemas-teste correspondem às respectivas soluções inteiras ótimas desses problemas.

A Tabela 4.3 mostra os tempos médios demandados (em segundos) pelo CPLEX para resolver as relaxações lineares das formulações matemáticas apresentadas nas Seções 4.1–4.4 aplicadas aos conjuntos de problemas-teste. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste com a mesma quantidade de tarefas. O carácter "–" diz que o CPLEX não foi capaz de resolver a respectiva relaxação linear de pelo menos um dos 16 problemas-teste dentro de uma hora de execução.

De acordo com a Tabela 4.3, os respectivos tempos médios demandados pelo CPLEX ao utilizar as formulações VP, VP', VPI, VPIA e VPS foram semelhantes e sempre inferiores a 2 segundos. Por outro lado, as relaxações lineares IMR e IMN foram as que exigiram maiores tempos médios do CPLEX. Os tempos médios requeridos pelo CPLEX com as relaxações lineares das formulações binárias BR e BN foram significativamente inferiores que os respectivos tempos médios requeridos com as formulações IMR e IMN. Dentre as relaxações lineares das

Tabela 4.3: Tempos médios demandados pelo CPLEX com as relaxações lineares das formulações apresentadas nas Seções 4.1–4.4 (em segundos).

n	VP	VP'	VPI	VPIA	VPS	IMR	IMN	BR	BN	B1	B2
06	0,04	0,04	0,04	0,04	0,04	0,16	0,27	0,22	0,41	0,25	0,23
07	0,05	0,05	0,05	0,06	0,06	0,30	0,63	0,39	0,78	0,47	0,45
08	0,07	0,07	0,07	0,07	0,07	0,43	1,20	0,57	1,09	0,67	0,68
09	0,11	0,11	0,11	0,11	0,12	0,84	1,98	0,92	1,76	1,01	1,04
10	0,14	0,14	0,14	0,14	0,16	1,61	3,21	1,59	3,03	1,82	1,92
11	0,20	0,20	0,20	0,20	0,22	3,07	17,51	1,97	3,92	2,17	2,45
12	0,31	0,31	0,31	0,32	0,34	23,70	5,80	2,62	5,04	2,83	3,19
13	0,39	0,39	0,39	0,39	0,42	22,26	17,31	3,33	6,61	3,66	4,80
14	0,50	0,50	0,50	0,50	0,56	5,35	10,42	4,92	11,35	6,12	7,85
15	0,75	0,75	0,75	0,75	0,81	139,06	237,25	5,61	12,24	7,35	9,89
16	0,65	0,65	0,65	0,65	0,74	304,25	459,58	6,41	15,61	8,41	10,89
17	0,91	0,91	0,91	0,93	1,02	1073,41	240,55	8,45	19,32	11,43	15,86
18	1,19	1,18	1,18	1,19	1,25	1792,10	72,68	10,82	26,25	15,26	22,35
19	1,51	1,51	1,51	1,52	1,60	1097,92	696,41	12,56	29,53	16,32	24,97
20	1,68	1,68	1,68	1,69	1,78	–	–	14,74	37,56	23,81	32,44

formulações baseadas em variáveis indexadas no tempo, os respectivos menores tempos médios demandados pelo CPLEX acorreram com a formulação BR. Por fim, os tempos médios requeridos pelo CPLEX com a relaxação linear da formulação B2, que foi a que proporcionou ao CPLEX encontrar soluções de menores *gap*'s médios entre as relaxações lineares das formulações baseadas em variáveis indexadas no tempo, foram menores ou iguais a 32,44 segundos.

Apesar de não ser mostrado na Tabela 4.3, o limite de tempo de uma hora foi atingido pelo CPLEX em 3 problemas-teste de 17 tarefas, 7 problemas-teste de 18 tarefas e 3 problemas-teste de 19 tarefas ao ser utilizado a relaxação linear da formulação IMR. Já ao se utilizar a relaxação linear da formulação IMN, o CPLEX atingiu o limite de tempo em 1 problema-teste de 15 tarefas, 2 problemas-teste de 16 tarefas, 1 problema-teste de 17 tarefas e 3 problemas-teste de 19 tarefas.

4.8.2 Limites inferiores obtidos com os algoritmos de separação propostos na Seção 4.7

O algoritmo de planos-de-corte descrito no Algoritmo 4.6 foi utilizado a fim de construir limites inferiores para o SMSPETP-SDS com base nos algoritmos de separação propostos na Seção 4.7. A família de restrições F, utilizada para construir o limite inferior, o respectivo algoritmo de separação para essa família e a violação mínima considerada (δ) são dados de entrada do Algoritmo 4.6.

No Algoritmo 4.6, PPM representa um problema de programação matemática que é atualizado iterativamente por meio de inclusões e exclusões de restrições. O PPM inicial é dado pela formulação PPM_0, definida pelas equações (4.74)–(4.76).

$$(PPM_0) \qquad \min \sum_{x \in I} \sum_{h \in H_x} g_x(h) \cdot l_{xh} \tag{4.74}$$

$$\text{s.a.} \qquad \sum_{h \in H_x} l_{xh} = 1 \quad \forall\, x \in I \tag{4.75}$$

$$l_{xh} \in [0, 1] \quad \forall\, x \in I \text{ e } \forall\, h \in H_x \tag{4.76}$$

Algoritmo 4.6 Limite inferior construído com uma dada família de restrições F e um dado δ.

$PPM \leftarrow PPM_0$;
$l^* \leftarrow$ solução do PPM;
Resolva o problema de separação da família F para l^* e δ;
enquanto *forem encontradas restrições da família* F *violadas por* l^* **faça**
  Adicione as restrições encontradas ao PPM corrente;
  $l^* \leftarrow$ solução do PPM corrente;
  Retire do PPM corrente as restrições satisfeitas por l^* com folga;
  Resolva o problema de separação da família F para l^* e δ;
fim
Retorne l^*;

A expressão (4.74), em que $g_x(h) = \alpha_x \cdot \max(E_x - h - P_x, 0) + \beta_x \cdot \max(h + P_x - T_x, 0)$, representa a função objetivo do SMSPETP-SDS. As equações (4.75) dizem que cada tarefa deve ser executada uma única vez, ao passo que as restrições (4.76) consiste na relaxação linear das variáveis $l_{xh},\ \forall x \in I$ e $\forall h \in H_x$. Após a inicialização do PPM e resolução desse problema pelo CPLEX, resolve-se o problema de separação da família de restrições F associado à violação mínima δ para a solução do PPM corrente l^*. Enquanto forem encontradas restrições da família F violadas pela solução corrente l^*, iterativamente, adiciona-se as restrições encontradas ao PPM e atualiza-se a solução corrente l^*. Com o intuito de manter o PPM composto apenas pelas restrições satisfeitas por l^* na igualdade, sempre que l^* é atualizada, as restrições do PPM satisfeitas por l^* com folga são retiradas desse problema. Quando o algoritmo de separação da família de restrições F não encontrar restrições violadas pela solução corrente l^*, a construção do limite inferior para o problema é interrompida e a solução do PPM corrente é retornada a fim de calcular esse limite.

O conjunto de problemas-teste com 15 tarefas foi utilizado a fim de calibrar o parâmetro δ (violação mínima considerada) do Algoritmo 4.6. Foram experimentados os valores 1,0; 0,8; 0,6; 0,4; 0,2 e 0,1 para δ. Foram construídos limites inferiores com cada uma das Famílias 1–5, propostas na Seção 4.6. Para resolver os problemas de separação dessas famílias, empregou-se os respectivos algoritmos de separação propostos na Seção 4.7. Os resultados obtidos são apresentados na Tabela 4.4. Nessa tabela, a primeira coluna mostra os valores de δ experimentados. Para cada valor de δ, as colunas "$\overline{gap}$" e "$\overline{\text{tempo}}$" apresentam, respectivamente, as médias dos *gap*'s dos limites inferiores obtidos (em porcentagem) e as médias dos tempos demandados (em segundos) ao se executar o Algoritmo 4.6 com as respectivas famílias de restrições e os respectivos problemas-teste com 15 tarefas.

Tabela 4.4: Resultados obtidos ao se aplicar o Algoritmo 4.6 com as Famílias 1–5 e diferentes valores de δ no problemas-teste com 15 tarefas.

δ	**Família 1**		**Família 2**		**Família 3**		**Família 4**		**Família 5**	
	$\overline{gap}$ (%)	$\overline{\text{tempo}}$ (s)	$\overline{gap}$ (%)	$\overline{\text{tempo}}$ (s)	$\overline{gap}$ (%)	$\overline{\text{tempo}}$ (s)	$\overline{gap}$ (%)	$\overline{\text{tempo}}$ (s)	$\overline{gap}$ (%)	$\overline{\text{tempo}}$ (s)
1,0	38,32	20,49	53,73	10,20	91,40	0,96	87,74	0,87	94,09	0,85
0,8	31,34	34,83	47,71	19,63	91,40	0,83	87,74	0,87	93,88	0,97
0,6	21,93	65,08	41,60	28,46	91,40	0,96	87,70	0,88	93,88	0,89
0,4	14,01	143,20	34,49	68,65	91,35	0,86	87,66	0,85	93,87	0,91
0,2	6,94	403,16	27,23	218,66	91,35	0,87	88,17	0,85	93,87	0,91
0,1	4,15	1041,04	24,60	537,26	91,35	0,88	88,18	0,92	93,87	0,95

De acordo com a Tabela 4.4, para todos os valores de δ testados, os respectivos menores

gap's médios são dos limites inferiores para o SMSPETP-SDS obtidos com a Família 1, seguidos dos respectivos *gap*'s médios dos limites inferiores obtidos com as Famílias 2, 4, 3 e 5, nessa ordem. Também para todos os valores de δ experimentados, os tempos médios demandados para construir limites inferiores com as Famílias 3, 4 e 5 foram inferiores a 1 segundo, enquanto as médias dos *gap*'s desses limites são maiores que 87%. Com as Famílias 1 e 2, quanto menor é o valor testado para δ, menor é a média dos *gap*'s dos limites inferiores construídos e maior é o tempo médio demandado.

Embora não seja apresentado na Tabela 4.4, para δ igual a 0,1, o tempo de execução do Algoritmo 4.6 aplicado a um dos problemas-teste foi de 6.619,83 segundos com a Família 1 e de 3.823,11 segundos com a Família 2. Visto que essas duas famílias de restrições foram as que proporcionam limites inferiores para o SMSPETP-SDS de melhor qualidade, com a intenção de reduzir esse tempo computacional, experimentou-se, também, uma estratégia com δ dinâmico.

O Algoritmo 4.7 detalha a estratégia com violação mínima (δ) dinâmica utilizada para construir limites inferiores para o SMSPETP-SDS com base em uma dada família de restrições F. Nessa estratégia, após a inicialização do PPM e resolução desse problema pelo CPLEX, o valor de δ é inicializado igual a 0,8. Resolve-se, então, o problema de separação da família de restrições F, considerando-se a violação mínima atual δ, para a solução do PPM corrente l^*. Enquanto forem encontradas restrições da família F violadas pela solução corrente l^*, adiciona-se as restrições encontradas ao PPM, atualiza-se a solução corrente l^*, retira-se do PPM as restrições satisfeitas por l^* com folga e resolve-se novamente o problema de separação da família de restrições F. Quando não forem encontradas restrições da família F violadas pela solução corrente l^* com o valor de δ atual, o valor de δ é dividido por 2. Se o novo valor de δ for maior ou igual a 0,1, o processo de resolução do problema de separação da família F para l^* e atualização do PPM é repetido. Caso contrário, isto é, se o novo valor de δ for menor que 0,1, então a construção do limite inferior para o problema é interrompida e a solução do PPM corrente é retornada a fim de calcular esse limite.

Algoritmo 4.7 Limite inferior construído com uma dada família de restrições F e δ dinâmico.

$PPM \leftarrow PPM_0$;
$l^* \leftarrow$ solução do PPM;
$\delta \leftarrow 0,8$;
Resolva o problema de separação da família F para l^* e δ;
enquanto $\delta \geq 0,1$ **faça**
  enquanto *forem encontradas restrições da família F violadas por* l^* **faça**
    Adicione as restrições encontradas ao PPM corrente;
    $l^* \leftarrow$ solução do PPM corrente;
    Retire do PPM corrente as restrições satisfeitas por l^* com folga;
    Resolva o problema de separação da família F para l^* e δ;
  fim
  $\delta \leftarrow \delta \div 2$;
fim
Retorne l^*;

Inicialmente, o Algoritmo 4.7 foi utilizado para construir limites inferiores com cada uma das Famílias 1–5 para os problemas-teste com 15 tarefas. Para resolver os problemas de separação dessas famílias, empregou-se os respectivos algoritmos de separação propostos na Seção 4.7. Os resultados obtidos são apresentados na Tabela 4.5. Nessa tabela, as colunas "$\overline{gap}$" e "$\overline{\text{tempo}}$" apresentam, respectivamente, as médias dos *gap*'s dos limites inferiores obtidos (em porcentagem) e as médias dos tempos demandados (em segundos) ao se executar o Algoritmo 4.7 com as respectivas famílias de restrições e os respectivos problemas-teste com 15 tarefas.

De acordo com as Tabelas 4.4 e 4.5, a estratégia com δ dinâmico proporcionou limites

Tabela 4.5: Resultados obtidos ao se aplicar o Algoritmo 4.7 com as Famílias 1–5 nos problemas-teste com 15 tarefas.

Família 1		**Família 2**		**Família 3**		**Família 4**		**Família 5**	
$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)
4,20	198,37	24,44	127,72	91,35	0,95	87,23	0,84	93,87	0,88

inferiores para o SMSPETP-SDS cujos *gap*'s médios são semelhantes aos respectivos *gap*'s médios dos limites inferiores obtidos com δ fixado em 0,1. Por outro lado, os tempos médios demandados para construir limites inferiores por meio da estratégia com δ dinâmico usando as Famílias 1 e 2 foram significativamente inferiores que os respectivos tempos médios exigidos para construir limites inferiores usando a estratégia com δ fixado em 0,1. Para a Família 1, por exemplo, a estratégia com δ dinâmico demandou tempo médio igual a 198,37 segundos, ao passo que a estratégia com δ fixado em 0,1 exigiu tempo médio igual a 1.041,04 segundos. Os tempos médios demandados para construir limites inferiores por meio da estratégia com δ dinâmico com as Famílias 3, 4 e 5 foram próximos dos respectivos tempos médios demandados por meio da estratégia com δ fixo, isto é, inferiores a 1 segundo.

Uma vez que a estratégia com δ dinâmico se mostrou a melhor estratégia testada nos problemas-teste com 15 tarefas, essa estratégia foi utilizada, também, para construir limites inferiores para os demais problemas-teste com até 20 tarefas. Os resultados obtidos são apresentados na Tabela 4.6. Nessa tabela, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste com a mesma quantidade de tarefas. Para cada conjunto de problemas, as colunas "$\overline{gap}$" e "$\overline{\text{tempo}}$" apresentam, respectivamente, as médias dos *gap*'s dos limites inferiores obtidos (em porcentagem) e as médias dos tempos demandados (em segundos) ao se executar o Algoritmo 4.7 com as respectivas famílias de restrições e os respectivos problemas-teste desse conjunto.

Tabela 4.6: Resultados obtidos ao se aplicar o Algoritmo 4.7 com as Famílias 1–5 nos problemas-teste com até 20 tarefas.

	Família 1		**Família 2**		**Família 3**		**Família 4**		**Família 5**	
n	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)
06	0,06	0,68	1,59	1,49	84,90	0,06	72,00	0,08	87,83	0,08
07	0,17	2,95	4,26	5,31	83,77	0,10	74,81	0,10	87,87	0,09
08	0,44	7,08	8,26	7,86	83,07	0,11	73,74	0,09	86,03	0,12
09	1,58	14,70	13,01	11,51	89,06	0,13	80,10	0,13	91,92	0,14
10	0,87	25,63	9,34	27,92	90,86	0,20	82,12	0,18	92,29	0,20
11	2,60	47,07	14,72	39,07	92,97	0,25	85,59	0,25	94,44	0,27
12	3,26	77,21	18,58	55,36	89,98	0,39	81,83	0,36	92,37	0,38
13	3,25	83,64	22,02	54,27	88,43	0,50	83,52	0,44	90,95	0,48
14	2,16	153,44	17,84	94,49	91,16	0,58	85,64	0,56	92,87	0,57
15	4,20	195,47	24,44	127,44	91,35	0,81	87,23	0,82	93,87	0,84
16	5,42	362,04	25,90	187,69	90,84	0,71	87,15	0,74	92,87	0,82
17	5,24	415,57	26,62	253,60	90,91	0,97	86,98	1,06	92,56	1,07
18	4,22	516,20	24,91	279,38	92,66	1,26	88,95	1,33	93,84	1,44
19	4,23	726,58	25,83	397,62	92,51	1,61	89,85	1,63	94,06	1,70
20	6,34	887,28	26,65	558,24	93,51	1,79	90,08	1,86	94,87	1,99

Conforme a Tabela 4.6, os respectivos menores *gap*'s médios são dos limites inferiores construídos com a Família 1, seguidos dos respectivos *gap*'s médios dos limites inferiores construídos com as Famílias 2, 4, 3 e 5, nessa ordem. A diferença entre os *gap*'s médios dos limites inferiores construídos com a Família 1 e os *gap*'s médios dos limites inferiores obtidos com a Família 2 é significativa. O mesmo acontece com a diferença entre os respectivos *gap*'s médios dos limites inferiores construídos com as Famílias 2 e 4. Os respectivos maiores tempos médios demandados também foram observados ao se utilizar a Família 1, seguidos dos respectivos tempos médios requeridos com a Família 2. Os tempos médios demandados com as Famílias 3, 4 e 5 foram menores que 2 segundos. No entanto, os *gap*'s médios dos limites inferiores construídos com essas famílias de restrições foram maiores ou iguais a 72,00%.

Por último, utilizou-se uma estratégia com violação mínima dinâmica e que faz uso das cinco famílias de restrições propostas na Seção 4.6 para gerar limites inferiores para os problemas-teste com até 20 tarefas. A estratégia utilizada é baseada na rotina de busca local Descida em Vizinhança Variável, *Variable Neighborhood Descent* – VND (Hansen & Mladenović, 2001) e pode ser representada pelo Algoritmo 4.8.

Algoritmo 4.8 Limite inferior construído com as Famílias 1–5.

$PPM \leftarrow PPM_0$;
$l^* \leftarrow$ solução do PPM;
$\delta \leftarrow 0,8$;
enquanto $\delta \geq 0,1$ **faça**
  $i \leftarrow 1$;
  enquanto $i \leq 5$ **faça**
    Resolva o problema de separação da i-ésima família de restrições para l^* e δ;
    se *forem encontradas restrições violadas por* l^* **então**
      Adicione as restrições encontradas ao PPM corrente;
      $l^* \leftarrow$ solução do PPM corrente;
      Retire do PPM corrente as restrições satisfeitas por l^* com folga;
      $i \leftarrow 1$;
    senão
      $i \leftarrow i + 1$;
    fim
  fim
  $\delta \leftarrow \delta \div 2$;
fim
Retorne l^*;

No Algoritmo 4.8, as famílias de restrições propostas na Seção 4.6 estão ordenadas de acordo com a ordem crescente dos respectivos *gap*'s médios apresentados na Tabela 4.6, isto é, Famílias 1, 2, 4, 3 e 5, nessa ordem. Após a inicialização do PPM e resolução desse problema pelo CPLEX, o valor de δ é inicializado igual a 0,8. Enquanto δ for maior ou igual a 0,1, iterativamente, resolve-se o problema de separação da primeira família de restrições para a solução do PPM corrente l^*, considerando-se a violação mínima atual δ. Se forem encontradas restrições da primeira família violadas pela solução corrente l^*, essas restrições são adicionadas ao PPM e a solução l^* é, então, atualizada. Caso contrário, resolve-se o problema de separação da segunda família de restrições. Se forem encontradas restrições da segunda família violadas pela solução corrente l^*, essas restrições são adicionadas ao PPM, a solução l^* é atualizada e volta-se à resolução do problema de separação associado à primeira família de restrições. Caso contrário, passa-se para o problema de separação da terceira família de restrições. Esse procedimento é repetido até que seja resolvido o problema de separação associado à quinta família de restrições. Sempre que forem encontradas restrições de uma dada família violadas

por l^*, essas restrições são adicionadas ao PPM, a solução l^* é atualizada e volta-se a resolver o problema de separação associado à primeira família de restrições. Por outro lado, se não for encontrada nenhuma restrição violada por l^* em uma dada família, resolve-se o problema de separação da família seguinte. Se não for encontrada nenhuma restrição por l^* na quinta família, considerando-se o valor de δ corrente, o valor de δ é, então, reduzido à metade e a busca por restrições violadas por l^* recomeça a partir da primeira família de restrições. A construção do limite inferior para o problema é interrompida quando δ atingir um valor menor que 0,1. Nesse caso, a solução do PPM corrente é retornada e fornece um limite inferior para o problema. Assim como nos Algoritmo 4.6 e 4.7, a fim de manter o PPM composto apenas pelas restrições satisfeitas por l^* na igualdade, sempre que l^* é atualizada, as restrições do PPM satisfeitas por l^* com folga são retiradas desse problema também no Algoritmo 4.8.

A Tabela 4.7 apresenta os resultados obtidos ao se utilizar a estratégia com δ dinâmico e que faz uso das Famílias 1–5 para gerar limites inferiores para os problemas-teste com até 20 tarefas. Visto que as relaxações lineares das formulações matemática VP' e B2 foram as que proporcionaram limites inferiores de menores *gap*'s médios nos resultados apresentados na Seção 4.8.1, a fim de facilitar uma comparação, os resultados obtidos com as relaxações lineares dessas duas formulações são repetidos nessa tabela.

Na Tabela 4.7, a primeira coluna indica o número de tarefas em cada conjunto de 16 problemas-teste com a mesma quantidade de tarefas. Para cada conjunto de problemas, as colunas "$\overline{gap}$" e "$\overline{\text{tempo}}$" da coluna "Famílias 1–5" apresentam, respectivamente, as médias dos *gap*'s dos limites inferiores obtidos (em porcentagem) e as médias dos tempos demandados (em segundos) ao se executar o Algoritmo 4.8 com as respectivas famílias de restrições e os respectivos problemas-teste desse conjunto. Já as colunas "$\overline{gap}$" e "$\overline{\text{tempo}}$" das colunas "VP'" e "B2" apresentam, respectivamente, as médias dos *gap*'s dos limites inferiores obtidos (em porcentagem) e as médias dos tempos demandados (em segundos) ao se utilizar o CPLEX para resolver as respectivas relaxações lineares dessas formulações matemáticas aplicadas aos problemas dos respectivos conjuntos de problemas-teste com o mesmo número de tarefas.

Tabela 4.7: Resultados obtidos ao se aplicar o Algoritmo 4.8 nos problemas-teste com até 20 tarefas.

	VP'		**B2**		**Famílias 1–5**	
n	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)	$\overline{gap}$ (%)	**tempo** (s)
06	73,66	0,04	29,64	0,23	0,00	0,65
07	63,03	0,05	45,65	0,45	0,02	2,79
08	66,69	0,07	50,93	0,68	0,24	7,15
09	59,19	0,11	54,15	1,04	1,29	14,88
10	66,08	0,14	55,30	1,92	0,46	26,66
11	70,01	0,20	60,56	2,45	1,90	50,40
12	62,11	0,31	65,43	3,19	2,21	89,28
13	64,36	0,39	59,85	4,80	2,80	94,84
14	68,42	0,50	61,69	7,85	1,71	171,17
15	68,09	0,75	67,32	9,89	3,18	231,52
16	64,47	0,65	69,25	10,89	4,94	372,63
17	63,22	0,91	70,25	15,86	4,88	470,41
18	73,11	1,18	66,81	22,35	3,92	578,00
19	64,85	1,51	69,18	24,97	3,79	829,96
20	68,48	1,68	72,17	32,44	5,89	1031,59

De acordo com a Tabela 4.5, os tempos médios demandados para construir os limites inferiores com as Famílias 1–5 foram sempre superiores que os respectivos tempos médios demandados para resolver as relaxações lineares das formulações VP' e B2. No entanto, os *gap*'s médios dos limites inferiores proporcionados pelo Algoritmo 4.8 são significativamente inferiores que os respectivos *gap*'s médios obtidos com as relaxações lineares das formulações VP' e B2. Os *gap*'s médios dos limites inferiores obtidos com as relaxações lineares das formulações VP' e B2 são superiores a 59% e 29%, respectivamente, ao passo que os *gap*'s médios construídos por meio das Famílias 1–5 são inferiores a 6%. O *gap* médio dos limites inferiores obtidos com as Famílias 1–5 para os problemas-teste com 6 tarefas são iguais a 0%, ou seja, o Algoritmo 4.8 encontrou as respectivas soluções inteiras ótimas desses problemas. Apesar de não ser mostrado na Tabela 4.2, o Algoritmo 4.8 encontrou as respectivas soluções inteiras ótimas de um total de 87 problemas-teste, entre eles, um problema-teste de 20 tarefas.

4.9 Conclusões do capítulo

Neste capítulo foram apresentadas várias formulações matemáticas para o SMSPETP-SDS. Os parâmetros de entrada das formulações foram definidos com base no horizonte de planejamento para a execução das tarefas proposto na Seção 2.3.

As duas primeiras formulações apresentadas são baseadas em variáveis de precedência, denominadas VP e VP'. A formulação VP foi proposta por Bustamante (2007), enquanto a formulação VP' consiste na formulação VP acrescida de uma família de restrições proposta neste capítulo. Em seguida, foram propostas duas formulações baseadas em variáveis de precedência imediata para o SMSPETP-SDS, denominadas VPI e VPIA. A formulação VPIA não faz uso das constantes suficientemente grandes presentes na formulação VPI. Logo depois, foi proposta uma formulação baseada em variáveis de posição na sequência para o problema, denominada VPS. Também foram apresentadas seis formulações baseadas em variáveis indexadas no tempo para o SMSPETP-SDS, denominadas IMR, IMN, BR, BN, B1 e B2. Somente a formulação IMR não foi proposta pela primeira vez neste trabalho. A formulação IMR é uma formulação linear inteira mista proposta por Rosa & Souza (2009), ao passo que a formulação IMN é obtida da formulação IMR por meio da substituição de um conjunto de restrições por outro mais forte. Por outro lado, as formulações BR e BN são obtidas das formulações IMR e IMN, respectivamente, pela eliminação das variáveis não binárias. As formulações B1 e B2 também são binárias, mas possuem um número de restrições aproximadamente igual à metade do número de restrições contidas nas formulações BR e BN. Por último, são propostas cinco novas famílias de restrições válidas para as formulações indexadas no tempo, bem como algoritmos de separação para essas famílias.

O otimizador CPLEX foi utilizado para resolver as relaxações lineares das formulações matemáticas apresentadas aplicadas aos problemas-teste com até 20 tarefas (vide Seção 2.2). As melhores soluções foram encontradas com as formulações lineares B2 e VP'. No entanto, a formulação B2 obteve melhores soluções para um maior número de problemas. O CPLEX obteve soluções parecidas com as relaxações lineares das formulações VP, VPI, VPS, IMR e IMN. A formulação VPIA obteve soluções sensivelmente melhores que as respectivas soluções obtidas com a formulação VPI. As relaxações lineares das formulações IMR e IMN foram as que exigiram maiores tempos para serem resolvidas.

A principal contribuição deste capítulo é a proposição das cinco famílias de restrições válidas para as formulações baseadas em variáveis indexadas no tempo do SMSPETP-SDS. As heurísticas de separação propostas para essas famílias também foram utilizadas para construir limites inferiores para os problemas-teste com até 20 tarefas. Os limites inferiores construídos com essas heurísticas são significativamente melhores que as respectivas soluções obtidas com as relaxações lineares das formulações matemáticas apresentadas neste trabalho. Embora os

tempos demandados para gerar tais limites inferiores sejam maiores que os respectivos tempos exigidos pelo CPLEX para resolver as relaxações lineares que obtiveram as melhores soluções (formulações B2 e VP'), os limites inferiores construídos são próximos das soluções inteiras ótimas dos respectivos problemas, chegando a corresponder às respectivas soluções inteiras ótimas de muitos problemas.

É importante observar que as restrições propostas para as formulações indexadas no tempo que representam o SMSPETP-SDS podem ser utilizadas, também, em formulações indexadas no tempo de diversos tipos de problemas de programação de tarefas envolvendo tempos de preparação da máquina dependentes da sequência de execução das tarefas.

Capítulo 5

Conclusões finais e trabalhos futuros

Neste capítulo são apresentadas as conclusões finais deste trabalho (Seção 5.1), bem como propostas para trabalhos futuros (Seção 5.2). Além disso, são apresentados os trabalhos oriundos desta pesquisa que foram publicados, ou aceitos para publicação, em eventos científicos e/ou periódicos especializados (Seção 5.3).

5.1 Conclusões finais

Este trabalho tratou o problema de programação de tarefas em uma máquina com janelas de conclusão distintas e tempos de preparação da máquina dependentes da sequência de execução das tarefas (SMSPETP-SDS). O objetivo é minimizar a soma ponderada das antecipações e atrasos na conclusão das tarefas. O caso particular com tempos de preparação da máquina independentes da sequência de execução das tarefas, denotado por SMSPETP-SIS, também foi tratado.

Inicialmente, a fim de construir bons limites superiores para o SMSPETP-SDS, tal problema foi tratado por meio de procedimentos heurísticos. Assim como nos trabalhos da literatura que abordam o problema dessa maneira, o SMSPETP-SDS foi dividido em dois subproblemas: determinar a melhor programação de uma dada sequência de tarefas, considerando-se a possibilidade de inserção de tempos ociosos entre a execução de tarefas consecutivas; e determinar uma sequência de tarefas que, associada à sua programação ótima, minimize a soma das penalidades geradas pelas antecipações e atrasos das tarefas.

Uma das contribuições mais importantes deste trabalho é o desenvolvimento de um algoritmo determinístico e de complexidade computacional $O(n^2)$, denotado por AAOTO (Algoritmo de alocação ótima de tempos ociosos), que resolve o problema de programação ótima de uma dada sequência de tarefas. Além disso, o estudo da complexidade computacional do algoritmo da literatura até então utilizado para esse fim é, também, uma contribuição desta tese. Provado que a complexidade do algoritmo de programação ótima de uma dada sequência de tarefas da literatura é $O(n^3)$, propôs-se uma adaptação para ele de forma a reduzir sua complexidade a $O(n^2)$.

Para resolver o problema de sequenciamento das tarefas do SMSPETP-SDS foram propostos três algoritmos heurísticos baseados na metaheurística GVNS (*General Variable Neighborhood Search*), denotados por GVNS$_r$, GVNS$_f$ e GVNS$_{rf}$. Propôs-se, também, um algoritmo exato de enumeração implícita para o SMSPETP-SIS. Tal algoritmo de enumeração implícita faz uso do algoritmo de programação ótima proposto, bem como de resultados teóricos desenvolvidos exclusivamente para o SMSPETP-SIS.

Experimentos computacionais feitos com um conjunto de problemas-teste gerado conforme parâmetros usuais da literatura mostraram que o algoritmo de programação ótima proposto é mais rápido que o algoritmo até então utilizado para esse fim. Os algoritmos heurísticos

propostos se mostraram mais eficientes que o melhor algoritmo de Ribeiro *et al.* (2010), pois foram capazes de encontrar soluções de melhor qualidade e em menor tempo computacional que esse algoritmo. Quando comparado com o melhor algoritmo de Rosa *et al.* (2017), os algoritmos propostos encontraram soluções de melhor qualidade para muitos problemas-teste e apenas o algoritmo GVNS$_f$ exigiu maior tempo computacional que o algoritmo de Rosa *et al.* (2017). Por outro lado, o algoritmo de enumeração implícita desenvolvido mostrou-se uma boa alternativa para a resolução exata do SMSPETP-SIS de dimensões menores, sendo capaz de resolver problemas-teste em tempo computacional muito inferior ao tempo exigido pelo otimizador CPLEX associado a uma formulação matemática da literatura.

Em um segundo momento, com o objetivo de definir bons limites inferiores para o problema, foram desenvolvidas várias formulações matemáticas para o SMSPETP-SDS. Um horizonte de planejamento para a execução de cada tarefa foi proposto a fim de ser utilizado na determinação dos parâmetros de entrada das formulações matemáticas para o problema. Também foram propostas novas famílias de restrições válidas para as formulações baseadas em variáveis indexadas no tempo, bem como algoritmos de separação para essas famílias.

Experimentos computacionais realizados com problemas-teste de até 20 tarefas mostraram que os limites inferiores construídos com os algoritmos de separação propostos são significativamente melhores que as respectivas soluções obtidas com as relaxações lineares das formulações matemáticas apresentadas neste trabalho. Apesar de os tempos demandados para gerar esses limites inferiores serem maiores que os respectivos tempos exigidos pelo otimizador CPLEX para resolver as relaxações lineares que obtiveram as melhores soluções, os limites inferiores construídos estão muito mais próximos das respectivas soluções inteiras ótimas dos problemas. Na verdade, os algoritmos de separação desenvolvidos permitiram encontrar as respectivas soluções ótimas de muitos dos problemas-teste experimentados.

As restrições propostas para as formulações baseadas em variáveis indexadas no tempo que representam o SMSPETP-SDS constituem uma contribuição de destaque deste trabalho pois, além de essas restrições permitirem encontrar bons limites inferiores para o SMSPETP, elas são válidas, também, para as formulações indexadas no tempo de diversas classes de problemas de programação de tarefas que consideram os tempos de preparação da máquina dependentes da sequência de execução das tarefas. Sendo assim, as novas famílias de restrições propostas para as formulações baseadas em variáveis indexadas no tempo podem ser incorporadas em estratégias que ajudem na resolução dessas formulações.

O algoritmo proposto AAOTO se mostrou a melhor estratégia experimentada para resolver o problema de programação de uma dada sequência de execução das tarefas do SMSPETP-SDS, e que o algoritmo proposto GVNS$_f$ se mostrou a melhor estratégia para prover limites superiores para o problema, ao passo que os melhores limites inferiores foram construídos por meio da estratégia que faz uso das novas famílias de restrições desenvolvidas neste trabalho.

5.2 Trabalhos futuros

Como trabalhos futuros, sugere-se utilizar o algoritmo de alocação ótima de tempos ociosos em uma data sequência de execução das tarefas do SMSPETP proposto neste trabalho em novos algoritmos para o problema.

Diante dos altos tempos demandados pelos algoritmos para resolver o SMSPETP, sugere-se estudar outras estratégias de redução do espaço de busca, bem como estratégias de processamento paralelo, para os algoritmos de resolução desse problema. Outra sugestão é estudar estratégias que permitam interromper o algoritmo de alocação ótima antes do posicionamento de todas as tarefas da sequência, tão logo seja identificado que a dada sequência não levará a uma solução de função objetivo melhor que aquele da solução corrente. Isso pode promover uma redução do custo computacional, em especial no caso de problemas maiores.

Outra sugestão de trabalhos futuros é o desenvolvimento de algoritmos para o SMSPETP que não dividam o problema em dois subproblemas, ou seja, que resolvam os problemas de sequenciamento e de programação das tarefas simultaneamente. Embora seja uma abordagem possível, não foram encontrados na literatura trabalhos que utilizam essa abordagem no SMSPETP.

Visto que as novas famílias de restrições válidas propostas neste trabalho para as formulações matemáticas que representam o SMSPETP permitiram construir bons limites inferiores para o problema, sugere-se, também, desenvolver novas restrições válidas para as formulações matemáticas que representam o SMSPETP, bem como desenvolver novos algoritmos de separação para as famílias de restrições válidas para as formulações baseadas em variáveis indexadas no tempo propostas neste trabalho. Ademais, sugere-se desenvolver estratégias de resolução inteira do SMSPETP que façam uso das formulações matemáticas e das famílias de restrições propostas neste trabalho.

Por último, sugere-se tratar o SMSPETP por meio da técnica de programação por restrições. Programação por restrições é um poderoso paradigma para resolver problemas de otimização que recorre a uma variedade de técnicas de inteligência artificial, ciência da computação e pesquisa operacional (Rossi *et al.*, 2006). Apesar de ser uma técnica promissora para problemas de programação de tarefas, não foi encontrado na literatura nenhum trabalho que trata o SMSPETP por meio dessa técnica.

5.3 Trabalhos derivados desta pesquisa

A seguir são listados os trabalhos gerados pela presente pesquisa.

1. Título: **Um Algoritmo *Branch-and-Bound* para o Problema de Sequenciamento em Uma Máquina com Penalidades por Antecipação e Atraso da Produção**
   - Autor: Bruno Ferreira Rosa
   - Evento: XVIII Escola Latino-Iberoamericana de Verão em Pesquisa Operacional – ELAVIO 2014
   - Local: Areia/PB - Brasil
   - Período: 17 a 21 de fevereiro de 2014
   - Trabalho apresentado no evento

2. Título: **Alocação de Tempos Ociosos em Uma Dada Sequência de Produção com Janelas de Entrega**
   - Autores: Bruno Ferreira Rosa, Marcone Jamilson Freitas Souza e Sérgio Ricardo de Souza
   - Evento: XLVI Simpósio Brasileiro de Pesquisa Operacional – SBPO 2014
   - Local: Salvador/BA - Brasil
   - Período: 16 a 19 de setembro de 2014
   - Páginas: 1858-1869

3. Título: ***Exact approaches for the single machine scheduling problem with distinct time windows***

   - Autores: Bruno Ferreira Rosa, Philippe Yves Paul Michelon, Zacharie Ales, Marcone Jamilson Freitas Souza e Sérgio Ricardo de Souza
   - Evento: *17ème Conférence de la société Française de Recherche Opérationnelle et d'Aide à la Décision* – ROADEF 2016
   - Local: Compiègne – França
   - Período: 22 a 24 de fevereiro de 2016
   - Trabalho aceito para apresentação no evento

4. Título: **Formulações matemáticas para o problema de programação de tarefas com janelas de entrega e tempos de preparação da máquina**

   - Autores: Bruno Ferreira Rosa, Marcone Jamilson Freitas Souza, Sérgio Ricardo de Souza, Philippe Yves Paul Michelon e Zacharie Ales
   - Evento: XLVIII Simpósio Brasileiro de Pesquisa Operacional – SBPO 2016
   - Local: Vitória/ES - Brasil
   - Período: 27 a 30 de setembro de 2016
   - Páginas: 4140-4151

5. Título: ***Algorithms for job scheduling problems with distinct time windows and general earliness/tardiness penalties***

   - Autores: Bruno Ferreira Rosa, Marcone Jamilson Freitas Souza, Sérgio Ricardo de Souza, Moacir Felizardo de França Filho, Zacharie Ales e Philippe Yves Paul Michelon
   - Periódico: *Computers & Operations Research*
   - Ano: 2017
   - Volume: 81, Páginas: 203-215

6. Título: ***Algorithms based on VNS for solving the Single Machine Scheduling Problem with Earliness and Tardiness Penalties***

   - Autores: Bruno Ferreira Rosa, Marcone Jamilson Freitas Souza e Sérgio Ricardo de Souza
   - Evento: *5th International Conference on Variable Neighborhood Search* – VNS 2017
   - Local: Ouro Preto/MG - Brasil
   - Período: 02 a 04 de outubro de 2017
   - Trabalho fará parte de um número especial do periódico *Electronic Notes in Discrete Mathematics*

Referências Bibliográficas

Allahverdi, A. 2015. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, **246**(2), 345 – 378.

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. 1999. A review of scheduling research involving setup considerations. *Omega*, **27**(2), 219–239.

Allahverdi, A., Ng, C.T., Cheng, T.C.E., & Kovalyov, Mikhail Y. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, **187**(3), 985–1032.

Ascheuer, Norbert, Fischetti, Matteo, & Grötschel, Martin. 2001. Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut. *Mathematical Programming*, **90**(3), 475–506.

Avella, Pasquale, Boccia, Maurizio, Mannino, Carlo, & Vasilyev, Igor. 2016. Valid Inequalities for Time-indexed Formulations of the Runway Scheduling Problem. *In: CEUR Workshop Proceedings*.

Baker, Kenneth R., & Keller, Brian. 2010. Solving the single-machine sequencing problem using integer programming. *Computers & Industrial Engineering*, **59**(4), 730 – 735.

Baker, Kenneth R., & Scudder, Gary D. 1990. Sequencing with Earliness and Tardiness Penalties: A Review. *Oper. Res.*, **38**(1), 22–36.

Bustamante, L. M. 2007. *Minimização do custo de antecipação e atraso para o problema de sequenciamento de uma máquina com tempo de preparação dependente da sequência: aplicação em uma usina siderúrgica.* Dissertação de mestrado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte.

Carrano, E.G., Wanner, E.F., & Takahashi, R.H.C. 2011. A Multicriteria Statistical Based Comparison Methodology for Evaluating Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, **15**(6), 848–870.

Davis, J. Steve, & Kanet, John J. 1993. Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics (NRL)*, **40**(1), 85–101.

Feo, T. A., & Resende, M. G. C. 1995. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, **6**(2), 109–133.

França Filho, M. F. 2007. *GRASP e Busca Tabu aplicados a problemas de programação de tarefas em máquinas paralelas.* Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil.

Fry, T. D., Armstrong, R. D., & Blackstone, J. H. 1987. Minimizing Weighted Absolute Deviation in Single Machine Scheduling. *IIE Transactions*, **19**(4), 445–450.

Garey, M. R., Tarjan, R. E., & Wilfong, G. T. 1988. One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties. *Mathematics of Operations Research*, **13**(2), 330–348.

Glover, Fred, & Laguna, Manuel. 1997. *Tabu Search.* Norwell, MA, USA: Kluwer Academic Publishers.

Gomes Júnior, A. C., Carvalho, C. R. V., Munhoz, P. L. A., & Souza, M. J. F. 2007. Um método heurístico híbrido para a resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. *Pages 1649–1660 of: Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional.* . SOBRAPO, Fortaleza, Brasil.

Gupta, Skylab R., & Smith, Jeffrey S. 2006. Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, **175**(2), 722–739.

Gutin, Gregory, & Punnen, Abraham P. 2007. *The traveling salesman problem and its variations.* Vol. 12. Springer Science & Business Media.

Hansen, P., & Mladenović, N. 2003. Variable Neighborhood Search. *145-184 6 of:* Glover, F., & Kochenberger, G. A. (eds), *Handbook of Metaheuristics.* Kluwer Academic Publishers.

Hansen, P., Mladenović, N., & Pérez, J. A. M. 2008. Variable neighborhood search: methods and applications. *4OR: A Quarterly Journal of Operations Research*, **6**(4), 319–316.

Hansen, Pierre, & Mladenović, Nenad. 2001. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(3), 449–467.

Hansen, Pierre, & Mladenović, Nenad. 2005. Variable Neighborhood Search. *Pages 211–238 of:* Burke, EdmundK., & Kendall, Graham (eds), *Search Methodologies.* Springer US.

Hansen, Pierre, & Mladenović, Nenad. 2014. *Variable Neighborhood Search.* Boston, MA: Springer US. Pages 313–337.

Janiak, Adam, Janiak, Władysław A., Krysiak, Tomasz, & Kwiatkowski, Tomasz. 2015. A survey on scheduling problems with due windows. *European Journal of Operational Research*, **242**(2), 347 – 357.

Józefowska, Joanna. 2007. *Just-in-time Scheduling: Models And Algorithms For Computer And Manufacturing Systems.* International Series In Operations Research & Management Science. New York: Springer.

Kanet, John J., & Sridharan, V. 2000. Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, **48**(1), 99–110.

Kopanos, Georgios M., Laínez, José Miguel, & Puigjaner, Luis. 2009. An Efficient Mixed-Integer Linear Programming Scheduling Framework for Addressing Sequence-Dependent Setup Issues in Batch Plants. *Industrial & Engineering Chemistry Research*, **48**(13), 6346–6357.

Koulamas, Christos. 1996. Single-machine scheduling with time windows and earliness/tardiness penalties. *European Journal of Operational Research*, **91**(1), 190–202.

Lee, C. Y., & Choi, J. Y. 1995. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Computers & Operations Research*, **22**(8), 857–869.

Lourenço, Helena R., Martin, Olivier C., & Stützle, Thomas. 2003. Iterated Local Search. *Pages 320–353 of:* Glover, Fred, & Kochenberger, Gary. A. (eds), *Handbook of Metaheuristics.* International Series in Operations Research & Management Science, vol. 57. Springer US.

Maffioli, Francesco, & Sciomachen, Anna. 1997. A mixed-integer model for solving ordering problems with side constraints. *Annals of Operations Research*, **69**(0), 277–297.

Manne, A. S. 1960. On the Job-shop Scheduling Problem. *Operations Research*, **8**, 219–223.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. 1960. Integer Programming Formulation of Traveling Salesman Problems. *J. ACM*, **7**(4), 326–329.

Mladenović, N., & Hansen, P. 1997. Variable neighborhood search. *Computers & Operations Research*, **24**, 1097–1100.

Montgomery, D. C., & Runger, G. C. 2013. *Applied Statistics and Probability for Engineers.* 6th edn. Wiley.

Nemhauser, G. L., & Savelsbergh, M. W. P. 1992. A Cutting Plane Algorithm for the Single Machine Scheduling Problem with Release Times. *Pages 63–83 of:* Akgül, Mustafa, Hamacher, Horst W., & Tüfekçi, Süleyman (eds), *Combinatorial Optimization: New Frontiers in Theory and Practice.* Berlin, Heidelberg: Springer Berlin Heidelberg.

Nogueira, T. H., de Carvalho, C. R. V., & Ravetti, M. G. 2014. Analysis of mixed integer programming formulations for single machine scheduling problems with sequence dependent setup times and release dates. *Optimization Online – Submitted for 4OR: A Quarterly Journal of Operations Research.*

Penna, P. H. V., Souza, M. J. F., Gonçalves, F. A. C. A., & Ochi, L. S. 2012. Uma heurística híbrida para minimizar custos com antecipação e atraso do sequenciamento da produção em uma máquina. *Produção*, **22**, 766–777.

Pinedo, Michael L. 2012. *Scheduling: Theory, Algorithms, and Systems.* 4th edn. Springer New York.

Rabadi, Ghaith, Mollaghasemi, Mansooreh, & Anagnostopoulos, Georgios C. 2004. A Branch-and-bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common Due-date and Sequence-dependent Setup Time. *Computers & Operations Research*, **31**(10), 1727–1751.

Ribeiro, Fábio Fernandes, Souza, Marcone Jamilson Freitas, & de Souza, Sérgio Ricardo. 2010. An Adaptive Genetic Algorithm to the Single Machine Scheduling Problem with Earliness and Tardiness Penalties. *Pages 203–212 of:* da Rocha Costa, Antônio Carlos, Vicari, Rosa Maria, & Tonidandel, Flavio (eds), *Advances in Artificial Intelligence - SBIA 2010.* Lecture Notes in Computer Science, vol. 6404. Springer Berlin Heidelberg.

Rosa, B. F., & Souza, M. J. F. 2009. Uma Nova Formulação de Programação Matemática Indexada no Tempo para Uma Classe de Problemas de Sequenciamento em Uma Máquina. *Pages 2898–2909 of: Anais do XLI Simpósio Brasileiro de Pesquisa Operacional.* . SOBRAPO, Porto Seguro (BA).

Rosa, B. F., Souza, M. J. F., & de Souza, S. R. 2010. Uma heurística de redução do espaço de busca para uma classe de problemas de sequenciamento de tarefas em uma máquina. *Pages 3583–3590 of: Anais do XVIII Congresso Brasileiro de Automática.*

Rosa, Bruno Ferreira, Souza, Marcone Jamilson Freitas, de Souza, Sérgio Ricardo, de França Filho, Moacir Felizardo, Ales, Zacharie, & Michelon, Philippe Yves Paul. 2017. Algorithms for job scheduling problems with distinct time windows and general earliness/tardiness penalties. *Computers & Operations Research*, **81**, 203 – 215.

Rossi, F., van Beek, P., & Walsh, T. (eds). 2006. *Handbook of Constraint Programming.* Elsevier.

Sousa, Jorge P., & Wolsey, Laurence A. 1992. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, **54**(1), 353–367.

Souza, M. J. F. 2011. *Inteligência Computacional para Otimização.* Notas de aula. Disponível em: *www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.pdf*. Visualizado em: 17/03/2017.

Souza, M. J. F., Ochi, L. S., & Maculan Filho, N. 2008. Minimizing earliness and tardiness penalties on a single machine scheduling problem with distinct due windows and sequence-dependent setup times. *Pages 1–6 of: Proceedings of the ALIO/EURO 2008 Conference*, vol. 1.

Szwarc, Wlodzimierz, & Mukhopadhyay, Samar K. 1995. Optimal timing schedules in earliness-tardiness single machine sequencing. *Naval Research Logistics (NRL)*, **42**(7), 1109–1114.

Tanaka, Shunji. 2012. An Exact Algorithm for the Single-Machine Earliness-Tardiness Scheduling Problem. *Pages 21 – 40 of:* Ríos-Mercado, Roger Z., & Ríos-Solís, Yasmín A. (eds), *Just-in-Time Systems.* Springer Optimization and Its Applications, vol. 60. Springer New York.

van Eijl, C. A. 1995. *A polyhedral approach to the delivery man problem.* Memorandum COSOR 95-19. Eindhoven University of Technology.

Wagner, Harvey M. 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, **6**(2), 131–140.

Wan, Guohua, & Yen, Benjamin P.-C. 2002. Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research*, **142**(2), 271–281.

Wolsey, L. A. 1998. *Integer programming.* New York, NY, USA: Wiley-Interscience.

Yano, Candace Arai, & Kim, Yeong-Dae. 1991. Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal of Operational Research*, **52**(2), 167–178.

Apêndice

Testes de hipóteses

A seguir são descritos os dois testes de hipóteses utilizados para comparar as soluções médias obtidas por dois algoritmos para um dada problema. Ambos os testes de hipóteses são unilaterais, e o nível de significância adotado é $\gamma = 0,05$. O primeiro teste é um teste de hipótese paramétrico para duas amostras independentes (Montgomery & Runger, 2013), enquanto o segundo é um teste de aleatoriedade (Carrano *et al.*, 2011). Testes de aleatoriedade são métodos estatísticos não paramétricos, ou seja, eles não são baseados na condição de que as amostras sejam provenientes de populações com distribuição normal (Montgomery & Runger, 2013).

O teste de aleatoriedade utilizado é fundamentado em Carrano *et al.* (2011) e está descrito no Algoritmo 5.1. Nesse algoritmo, $A_1 = (a_1, a_2, \ldots, a_{n_1})$ e $A_2 = (b_1, b_2, \ldots, b_{n_2})$ são vetores que armazenam as soluções obtidas pelos algoritmo A_1 e A_2, respectivamente, aplicados ao problema-teste dado. O nível de significância do teste é dado por γ, com $0 < \gamma < 1$.

Algoritmo 5.1 Teste de aleatoriedade(A_1, A_2, γ)

$d \leftarrow \frac{1}{n_2}\sum_{i=1}^{n_2} b_i - \frac{1}{n_1}\sum_{i=1}^{n_1} a_i$;
$V \leftarrow (a_1, a_2, \ldots, a_{n_1}, b_1, b_2, \ldots, b_{n_2})$;
para $i = 1, 2, \ldots, 500$ **faça**
  $V' \leftarrow$ o vetor V embaralhado;
  $w_i \leftarrow \frac{1}{n_2}\sum_{j=1}^{n_2} v'_j - \frac{1}{n_1}\sum_{j=n_2+1}^{n_2+n_1} v'_j$; // $W = \{w_1, w_2, \ldots, w_{500}\}$
fim;
$p_{value}(d) \leftarrow P(W < d)$; `// Probabilidade de se escolher aleatoriamente, usando uma distribuição uniforme, um elemento de` W `menor que` d.
se $p_{value}(d) \geq 1 - \gamma$ **então**
  rejeita H_0 e aceita H_1;
fim

Dados dois algoritmos A_1 e A_2, as seguintes hipóteses (nula e alternativa) foram formuladas para comparar as médias das soluções obtidas por esses algoritmos aplicados a um dado problema-teste:

- Hipótese nula (H_0): as médias das soluções obtidas pelos algoritmos A_1 e A_2 são iguais;
- Hipótese alternativa (H_1): a média das soluções obtidas pelo algoritmo A_1 é menor que a média das soluções obtidas pelo algoritmo A_2.

Portanto, a hipótese nula afirma que, com o nível de significância adotado, não existem evidências estatísticas suficientes para afirmar que a média das soluções obtidas pelo algoritmo

A_1 é menor que a média das soluções obtidas pelo algoritmo A_2. Por outro lado, rejeitar a hipótese nula, e consequentemente aceitar a hipótese alternativa, significa que, com o nível de significância adotado, existem evidências estatísticas de que as soluções obtidas pelo algoritmo A_1 são, em média, melhores que as soluções obtidas pelo algoritmo A_2.