GUILHERME BAUMGRATZ FIGUEIROA

Orientador: Túlio Ângelo Machado Toffolo
Co-orientador: George Henrique Godim da Fonseca

HEURÍSTICAS MATEMÁTICAS PARA O PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO RELACIONADAS COM TEMPO DE PREPARO E SEQUÊNCIA DEPENDENTE

Ouro Preto
Dezembro de 2021

Universidade Federal de Ouro Preto
Departamento de Computação
Programa de Pós-Graduação em Ciência da Computação

HEURÍSTICAS MATEMÁTICAS PARA O PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO RELACIONADAS COM TEMPO DE PREPARO E SEQUÊNCIA DEPENDENTE

Proposta de dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

GUILHERME BAUMGRATZ FIGUEIROA

Ouro Preto
Dezembro de 2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

F475h Figueiroa, Guilherme Baumgratz.
Heurísticas matemáticas para o Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas com Tempo de Preparo e Sequência Dependente. [manuscrito] / Guilherme Baumgratz Figueiroa. - 2021.
55 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Túlio Ângelo Machado Toffolo.
Coorientador: Prof. Dr. George Henrique Godim da Fonseca.
Dissertação (Mestrado Acadêmico). Universidade Federal de Ouro Preto. Departamento de Computação. Programa de Pós-Graduação em Ciência da Computação.
Área de Concentração: Ciência da Computação.

1. Fix-and-optimize. 2. UPMSP. 3. Heurística Matemática. I. Fonseca, George Henrique Godim da. II. Toffolo, Túlio Ângelo Machado. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004

Bibliotecário(a) Responsável: Luciana De Oliveira - SIAPE: 1.937.800

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO
PROGRAMA DE POS-GRADUACAO EM CIENCIA DA COMPUTACAO

FOLHA DE APROVAÇÃO

Guilherme Baumgratz Figueiroa

Heurísticas matemáticas para o problema de escalonamento em máquinas paralelas não relacionadas com tempo de preparo e sequência dependente

Dissertação apresentada ao Programa de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Mestre em Ciência da Computação

Aprovada em 15 de dezembro de 2021

Membros da banca

Prof. Dr. Túlio Ângelo Machado Toffolo - Orientador - Universidade Federal de Ouro Preto
Prof. Dr. George Henrique Godim da Fonseca - Co-Orientador - Universidade Federal de Ouro Preto
Dr. Luciano Perdigão Cota - Instituto Tecnológico Vale
Prof. Dr. Puca Huachi Vaz Penna - Universidade Federal de Ouro Preto

Prof. Dr. Túlio Ângelo Machado Toffolo, orientador do trabalho, aprovou a versão final e autorizou seu depósito no Repositório Institucional da UFOP em 15/02/2022

Documento assinado eletronicamente por **Puca Huachi Vaz Penna**, **COORDENADOR(A) DE CURSO DE PÓS-GRADUACÃO EM CIÊNCIA DA COMPUTAÇÃO**, em 18/03/2022, às 12:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0295568** e o código CRC **C616003E**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: (31)3559-1641 - www.ufop.br

Dedico este trabalho a minha mãe, Suely, meu irmão, Caico, e para minha esposa, Thamiris, pois graças a eles que tive a força do início ao fim deste projeto.

Agradecimentos

Agradeço primeiramente a minha esposa, que em todo o tempo de mestrado passou, de namorada para noiva e de noiva para esposa, ela quem sempre me auxiliava quando precisava me dedicar mais tempo ao mestrado e me dava forças quando eu não tinha.
Agradeço também a minha família, mas principalmente minha mãe, Suely, meu irmão, Caico e minha prima/irmã, Ju, que sempre me animaram e me motivaram a ir até o final desse trabalho.
Agradeço também a minha República Bino, esta que me gerou vários momento de alegre e amizades que devo levar até o fim. Eles que também em deram suporte a auxílios, seja com texto, seja com momento de descontração.
Agradeço os meu orientador, Túlio, meu coorientador, George, por me ajudarem a trilhar esse caminho.
Agradeço também os meu colegas e amigos de serviço da Loggi que sempre estava me motivando a continuar em frente
Agradeço também a Nintendo pelo seu console e jogos para me distrair quando necessitava.

Resumo

O Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas considera um conjunto de tarefas e um conjunto de máquinas homogêneas ou heterogêneas que trabalham em paralelo. Todas as tarefas do conjunto devem ser processadas, sendo necessário escolher em qual máquina cada tarefa será executada. O objetivo é escalonar as tarefas nas máquinas de forma a minimizar o tempo total necessário para executar todas as tarefas, conhecido como *makespan*, dado pela máquina com maior tempo de processamento. Este trabalho estuda um caso do problema em que as tarefas são independentes, as máquinas heterogêneas e existe um tempo de preparo para execução de cada tarefa, que pode variar dependendo da sequência das tarefas e da máquina. Os principais modelos matemáticos propostos para o problema foram avaliados utilizando o conjunto de instâncias apresentado por Vallada e Ruiz (2011). Dentre os modelos e métodos analisados, o modelo proposto por Avalos-Rosales et al. (2015) e o algoritmo exato proposto por Fanjul-Peyro et al. (2019) obtiveram os melhores resultados tanto em termos de qualidade de solução quanto em termos de tempo de processamento. Assim, com o intuito de abordar instâncias maiores do problema, este trabalho propõe o uso de heurísticas matemáticas *Fix-And-Optimize* que utilizando os principais modelos disponíveis na literatura. As metodologias propostas consistem em decompor de forma heurística o problema por meio da fixação de um conjunto de tarefas e máquinas. Cada fixação resulta em um subproblema que pode ser resolvido por um modelo matemático ou método exato. Duas variações do algoritmos foram avaliadas, usando os modelos de Avalos-Rosales et al. (2015) e Fanjul-Peyro et al. (2019). Os resultados computacionais mostram que ambos algoritmos propostos obtêm valores próximos do melhor conhecido na literatura. Foram obtidas, ainda, diversas soluções melhores do que a melhor conhecida até então. Dentre as duas abordagens propostas, o algoritmo que utiliza o método de Fanjul-Peyro et al. (2019) para resolver subproblemas obteve os melhores resultados, sendo capaz de obter soluções melhores do que a melhor da literatura para 338 das 1000 instâncias de grande porte consideradas.

Palavras Chaves: *Fix-and-Optimize*, UPMSP, Heurística Matemática

Abstract

The Unrelated Parallel Machines Scheduling Problem considers a set of tasks and a set of homogeneous or heterogeneous machines that work in parallel. All tasks in the set must be processed, and it is necessary to choose on which machine each task will be executed. The objective is to schedule the tasks on the machines in order to minimize the total time needed to execute all the tasks, known as *makespan*, given by the machine with the most processing time. This work studies a case of the problem in which the tasks are independent, the machines are heterogeneous and there is a preparation time for the execution of each task, which can vary depending on the sequence of tasks and the machine. The main mathematical models proposed for the problem were evaluated using the set of instances presented by Vallada e Ruiz (2011). Among the models and methods analyzed, the model proposed by Avalos-Rosales et al. (2015) and the exact algorithm proposed by Fanjul-Peyro et al. (2019) obtained the best results both in terms of solution quality and in terms of processing time. Thus, in order to address larger instances of the problem, this work proposes the use of *Fix-And-Optimize* mathematical heuristics using the main models available in the literature. The proposed methodologies consist of heuristically decomposing the problem by setting a set of tasks and machines. Each fixation results in a sub-problem that can be solved by a mathematical model or exact method. Two variations of the algorithms were evaluated, using the models of Avalos-Rosales et al. (2015) and Fanjul-Peyro et al. (2019). The computational results show that both proposed algorithms obtain values close to the best known in the literature. In addition, several better solutions were obtained than the best known until then. Among the two proposed approaches, the algorithm that uses the Fanjul-Peyro et al. (2019) method to solve subproblems obtained the best results, being able to obtain better solutions than the best in the literature for 338 of the 1000 large instances considered.

Keywords: Fix-and-Optimize, UPMSP, Mathematical Heuristics

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

Ao longo das décadas, a área de Pesquisa Operacional (PO) vem apresentando grandes progressos. Cada vez mais surgem estudos para compreender melhor ou avaliar novas formas de abordar problemas em PO. Dentre os principais métodos utilizados, Programação Inteira Mista (*Mixed Integer Programming* - MIP) merece destaque, visto que tem sido aplicado para resolver inúmeros problemas, como o Problema de Escalonamento em Máquinas Paralelas Não Relacionadas abordado neste trabalho.

O Problema de Escalonamento em Máquinas considera um conjunto de tarefas, que podem ser relacionadas (ou seja, existe uma precedência entre as tarefas) ou independentes, que serão executadas por um conjunto de máquinas homogêneas ou heterogêneas trabalhando de forma paralela. Diferentes variantes do problema podem considerar diferentes objetivos, como diminuir o atraso das tarefas, diminuir o fluxo de tempo de cada máquina, reduzir o *makespan* (maior tempo de execução entre as máquinas), entre outros. Independente do objetivo e restrições consideradas, em geral é necessário que todas as tarefas sejam executadas pelas máquinas. O presente estudo considera o Problema de Escalonamento em Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependente da Sequência (*Unrelated Parallel Machine Scheduling Problem with Sequence Dependent and Setup Times* - UPMSP). O UPMSP consiste em distribuir um conjunto de tarefas a um conjunto de máquinas heterogêneas e paralelas de forma a minimizar o tempo total de execução, ou seja, minimizar o *makespan*. Neste problema, a alocação de uma tarefa a uma máquina é influenciada pelo tempo de execução na máquina, já que as máquinas podem requerer tempos diferentes para executar uma mesma tarefa. A alocação é influenciada também pelo tempo de preparo entre duas tarefas em uma máquina, dado pelo tempo de retirar uma tarefa e alocar uma nova tarefa. Assim como ocorre com o tempo de execução, o tempo de preparação também pode variar entre as diferentes máquinas.

O UPMSP pertence à classe $\mathcal{NP}$-Difícil (Lenstra et al., 1977). Por esta razão, diversos autores utilizam algoritmos heurísticos ou meta-heurísticos, como Rabadi et al. (2006), Vallada e Ruiz (2011) e Santos et al. (2016b), ou decomposições, como Tran e Beck (2012) e Avalos-

Rosales et al. (2015), para tratá-lo.

Neste trabalho são propostas heurísticas matemáticas para abordar o UPMSP, que consiste em decompor heurísticamente o problema em subproblemas de menor tamanho que podem ser resolvidos facilmente por métodos exatos. Como vários autores estudaram o UPMSP, os métodos exatos existentes na literatura foram avaliados utilizando as instâncias pequenas de Vallada e Ruiz (2011), com o objetivo de descobrir qual se adapta melhor à abordagem proposta nesse trabalho. Os melhores resultados foram obtidos a partir do modelo matemático proposto por Avalos-Rosales et al. (2015), que obteve os melhores limites (primais e duais) no menor tempo. Além deste modelo, avaliamos também o método proposto por Fanjul-Peyro et al. (2019), que utiliza um *branch-and-check* para resolver o UPMSP. O modelo e o método foram então utilizados para resolver os subproblemas obtidos a partir das decomposições heurísticas propostas.

A técnica de fixar parte do problema e otimizar o restante é conhecido como *Fix-and-Optimize* (FixOpt). Alguns trabalhos utilizaram este método para encontrar melhores resultados em um tempo menor em outros problemas como Santos et al. (2016a), Toffolo et al. (2016) e Holm et al. (2019). Nesta técnica, em geral é utilizado um modelo matemático para resolver os subproblemas. Para isso, algumas variáveis ficam com valores fixos, assim parte do modelo fica imutável, resultando em um subproblema mais simples de ser resolvido. Neste trabalho, são fixadas algumas tarefas e máquinas de forma que as tarefas livres podem ser alteradas de posição ou de máquina. Experimentos computacionais usando as instâncias de Vallada e Ruiz (2011) demonstram que os algoritmos propostos são competitivos com o estado-da-arte, sendo capazes inclusive de obter algumas soluções melhores do que as melhores conhecidas na literatura.

1.1 Objetivos

O objetivo principal do presente trabalho é propor heurísticas matemáticas para obter rapidamente soluções de boa qualidade para o UPMSP. As instâncias disponibilizadas por Vallada e Ruiz (2011) são utilizadas para comparar a abordagem proposta com aquelas encontradas na literatura.

1.1.1 Objetivos específicos

Os objetivos específicos do presente trabalho são:

- Pesquisar a literatura sobre heurísticas, meta-heurísticas e métodos exatos para o UPMSP;
- Estudar e implementar os métodos exatos propostos por outros autores;

- Estudar e propor heurísticas matemáticas para abordar instâncias do problema que atualmente nenhum método exato é capaz de solucionar;
- Avaliar diferentes estratégias e heurísticas matemáticas para o problema e determinar, dentre elas, a mais adequada para o UPMSP;
- Incorporar o método de *branch and check* proposto por Fanjul-Peyro et al. (2019) no algoritmo proposto.

1.1.2 Organização do trabalho

Este trabalho é composto por seis capítulos. No Capítulo 2 é definido o UPMSP, sua classificação, além de citar trabalhos relacionados. No Capítulo 3 são apresentadas as formulações propostas para o UPMSP, bem como os resultados de experimentos considerando tais formulações. O Capítulo 3 também apresenta e avalia o algoritmo *branch-and-check* proposto por Fanjul-Peyro et al. (2019). O Capítulo 4 descreve os algoritmos propostos por este trabalho. O Capítulo 5 apresenta os resultados e discussões dos experimentos com as instâncias propostas por Vallada e Ruiz (2011). Por fim, o Capítulo 6 apresenta as conclusões do trabalho e sugestões para trabalhos futuros.

Capítulo 2

O Problema de Escalonamento em Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependente da Sequência

Neste capítulo é apresentada a definição do Problema de Escalonamento em Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependente da Sequência (*Unrelated Parallel Machine Scheduling Problem With Sequence Dependent Setup Time* - UPMSP), a classificação do problema em relação a outros Problemas de Escalonamento em Máquinas e uma revisão dos trabalhos relacionados.

2.1 Definição do Problema

O UPMSP é formado por um conjunto de máquinas heterogêneas e um conjunto de tarefas, e deve-se alocar todas as tarefas nas máquinas disponíveis. Por serem máquinas heterogêneas, o tempo de execução de uma mesma tarefa pode ser diferente em cada máquina. O tempo de preparação entre duas tarefas altera de acordo com a máquina, portanto a ordem de cada tarefa em uma máquina altera o tempo de execução total desta máquina. Todas as máquinas começam não configuradas para receber a primeira tarefa, sendo assim é necessário prepará-las para a primeira tarefa. Como se trata de máquinas paralelas não relacionadas, cada máquina pode ter tempo de processamento diferente. A máquina com o maior tempo de processamento total define o valor do *makespan*, que deve ser minimizado.

Exemplo do UPMSP

Considere três máquinas heterogêneas e três tarefas quaisquer. O tempo de execução, dado em segundos, de cada tarefa em cada máquina é apresentado na Tabela 2.1. O tempo de preparação por máquina, também dado em segundos, é apresentado na Tabela 2.2 (a tarefa 0 é uma tarefa artificial usada para representar o tempo de preparação inicial da máquina).

Tabela 2.1: Tempo de execução de cada tarefa em cada máquina.

$Tarefa$	M_1	M_2	M_3
1	2	5	6
2	3	4	5
3	7	3	2

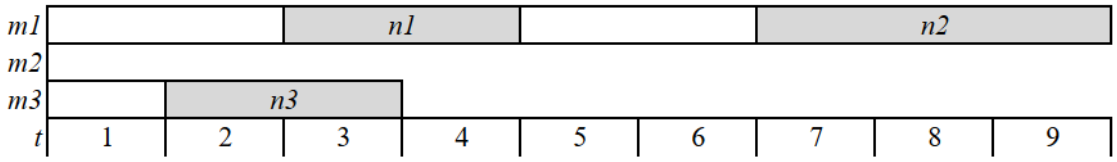
Tabela 2.2: Tempo de preparação de uma tarefa para outra nas máquinas.

M_1	1	2	3
0	2	3	4
1	0	2	5
2	5	0	3
3	3	1	0

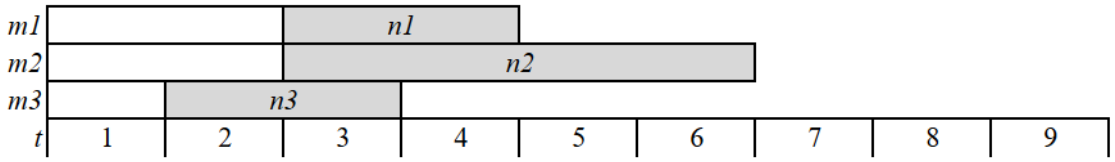
M_2	1	2	3
0	5	2	9
1	0	8	3
2	3	0	5
3	4	2	0

M_3	1	2	3
0	1	1	1
1	0	7	3
2	9	0	4
3	6	5	0

Um método para identificar uma solução é verificar as máquinas que requerem menos tempo para executar cada tarefa. Assim, as tarefas 1 e 2 seriam executadas pela máquina 1, a tarefa 3 seria executada pela máquina 3. A Figura 2.1 apresenta duas soluções distintas para esse mesmo problema. Nestas figuras, os blocos que contêm n_i representam cada tarefa, em que i é o número referente ao número da tarefa e os blocos vazios representam o tempo de preparação para a tarefa seguinte. A parte à baixo dessas figuras contém uma linha do tempo, em segundos. A Figura 2.1(a) apresenta uma solução gerada de forma aleatória. Esta solução tem o valor do *makespan* igual a 9 segundos. É possível notar que máquina 2 não é utilizada, sendo possível a execução de uma tarefa nessa máquina. Como a máquina 1 contém duas tarefas, é possível transferir uma tarefa para outra máquina. Foi escolhida a tarefa 2 por ter o menor tempo de preparação entre as tarefas 1 e 2, para a máquina 2, obtendo uma nova solução. Com essa alteração, o *makespan* reduziu seu valor de 9 para 6 unidades. Essa solução é apresentada na Figura 2.1(b). A Figura 2.1(b) apresenta uma nova solução, sendo esta a solução ótima para este caso. Neste caso do exemplo, existe somente esta solução ótima, mas em outros problemas pode existir mais de uma solução ótima.



(a) Uma solução do problema exemplo.



(b) Solução ótima do problema exemplo.

Figura 2.1: Duas soluções para o problema exemplo apresentado.

2.2 Classificação

Allahverdi (2015) apresentou a notação $\alpha|\beta|\gamma$, que descreve as características de Problemas de Escalonamento em Máquinas. O campo α descreve a relação entre as máquinas, se existe somente uma máquina, se são máquinas idênticas, máquinas heterogêneas, entre outras possibilidades, apresentadas na Tabela 2.3. O campo β descreve as restrições do problema, como por exemplo, se a máquina precisa ser preparada para receber uma tarefa, se existe uma ordem de precedência, entre outras características, mostradas na Tabela 2.4. Por fim, o campo γ descreve o objetivo do problema, por exemplo, reduzir o *makespan*, diminuir a latência entre as tarefas, diminuir o atraso das tarefas, entre outros objetivos, mostrados na Tabela 2.5.

O UPMSP, problema abordado pelo presente trabalho, é composto por máquinas heterogêneas (R), com tempo de preparação e sequência dependente (ST_{sd}) e o objetivo é minimizar o *makespan* (C_{max}). Portanto, segundo a notação $\alpha|\beta|\gamma$, o UPMSP é $R|ST_{sd}|C_{max}$.

2.3 Trabalhos relacionados

2.3.1 UPMSP

Allahverdi (2015) apresentou um *survey* dos principais estudos sobre o Problema de Escalonamento em Máquinas Paralelas (*Parallel Machine Scheduling Problem* - PMSP). O autor mostrou os principais artigos na área, apresentando as contribuições para cada valor de α, β e γ. Nessa seção, serão abordados os principais trabalhos relacionados ao PMSP com tempo de preparação dependente da sequência.

Lenstra et al. (1977) demonstraram que o PMSP utilizando duas máquinas idênticas e com o objetivo de minimizar o *makespan* ($P||C_{max}$) pertence à classe $\mathcal{NP}$-difícil. O UPMSP é uma generalização desse problema, já que o PMSP é uma especialização do UPMSP que

Tabela 2.3: Possíveis valores para o campo α. Fonte: Allahverdi (2015)

Notação	Descrição
1	Uma máquina simples.
P	Contém máquinas paralelas idênticas, ou seja, todas as máquinas executam as tarefas com o mesmo tempo.
Q	Contém máquinas paralelas uniformes, ou seja, cada máquina tem sua velocidade de execução, sendo o tempo para processamento uma tarefa igual ao custo de processar a tarefa divido pela velocidade da máquina.
R	Contém máquinas paralelas híbridas, ou seja, cada máquina contém tempo de execução de diferente para cada tarefa.
F_m	Flowshop com m máquinas: todas as tarefas têm o mesmo tempo de processamento.
FF_m	Flowshop com m máquinas flexíveis: todas as tarefas têm o mesmo tempo de processamento.
AF_m	Flowshop com m máquinas assembly: todas as tarefas têm o mesmo tempo de processamento.
J	Job shop : todas as tarefas executam no mesmo tempo.
FJ	Job shop flexível : todas as tarefas executam no mesmo tempo.
O	Open shop: pode existir tempo de processamento nulo (zero).

Tabela 2.4: Possíveis valores para o campo β. Fonte: Allahverdi (2015)

Notação	Descrição
ST_{si}	Sequência independente com tempo de preparação
SC_{sd}	Sequência independente com preparação de custo
ST_{sd}	Sequência dependente com tempo de preparação
$ST_{si,f}$	Sequência independente com tempo de preparação, utilizando família de tarefas
$ST_{sd,f}$	Sequência dependente com tempo de preparação, utilizando família de tarefas
$SC_{sd,f}$	Sequência dependente com preparação de tempo, utilizando família de tarefas
SC_{psd}	Sequência de passado dependente com preparação de tempo
$Prec$	Quando as tarefas têm uma ordem de precedência, geralmente representadas por um grafo orientado
r_j	Data de lançamento diferente de zero

considera somente máquinas iguais e os tempos de preparo iguais a zero. Portanto o UPMSP é, no mínimo, tão difícil quanto o PMSP de máquinas idênticas, e portanto também pertence à classe $\mathcal{NP}$-Difícil.

O UPMSP vem sendo estudado desde Guinet (1993), que realizou estudos sobre como melhorar a utilização de máquinas na área têxtil, focado nas indústrias francesas. Muitas indústrias têxteis que existiam antes da Segunda Guerra Mundial receberam várias máquinas de anos diferentes, sendo necessário otimizar a atribuição de tarefas a estas máquinas. Como as indústrias queriam velocidade e viabilidade para atender a demanda dos clientes, Guinet (1993) se propôs a estudar métodos para auxiliar a resolver o UPMSP. O autor apresentou uma formulação matemática para o problema, porém este problema contém diferenças em relação ao UPMSP, já que existe precedência entre as tarefas a serem executadas. Apesar

Tabela 2.5: Possíveis valores para o campo γ. Fonte: Allahverdi (2015)

Notação	Descrição
C_{max}	Tempo gasto para todas as máquinas terminarem de executar suas respectivas tarefas ou *makespan*
L_{max}	Latência máxima
T_{max}	Atraso máximo
D_{max}	Tempo máximo de entrega
TSC	Custo total de preparação
TST	Tempo total de preparação
TNS	Número total de preparação
$\sum F_j$	Total do fluxo de tempo
$\sum C_j$	Soma do tempo total de de cada máquina
$\sum E_j$	Total do tempo adiantado
$\sum T_j$	Total do tempo tardio
$\sum W_j$	Total do tempo em espera
$\sum U_j$	Total de números de tarefas tardias
$\sum w_j C_j$	Tempo total ponderado para finalizar as execuções em uma máquina
$\sum w_j F_j$	Total ponderado do fluxo de tempo
$\sum w_j U_j$	Total ponderado de números de tarefas tardias
$\sum w_j E_j$	Total ponderado do tempo adiantado
$\sum w_j T_j$	Total ponderado do tempo tardio
$\sum w_j W_j$	Total ponderado do tempo em espera
$\sum h(E_j)$	Total de penalidade por tarefas adiantadas
$\sum h(T_j)$	Total de penalidade por tarefas tardias
$TADC$	Diferença absoluta total no tempo para completar as tarefas

desta diferença, esta formulação foi utilizada como base para outros autores que propuseram formulações para o UPMSP.

Desde então, o UPMSP tem sido alvo de diversos estudos. Rabadi et al. (2006) estudaram um caso da indústria de treliças, que requerem configurações diferentes dependendo da sequência de fabricação e do tipo de máquina usada. Os autores apresentaram uma formulação matemática, baseada na formulação de Guinet (1993). Além disso, eles utilizaram a meta-heurística *Metaheuristic for Randomized Prority Search* (Meta-RaPS), proposta por DePuy et al. (2005) para o Problema do Caixeiro Viajante. Essa meta-heurística realiza procedimentos aleatórios baseados nos valores da ordem de preferência de cada tarefa, as restrições do problema e o quanto uma alteração melhora uma solução. Os autores criaram um conjunto de instâncias de teste que disponibilizaram online[1] e suas respectivas soluções.

Helal et al. (2006) e Arnaout et al. (2010) realizaram estudos sobre UPMSP, tendo como foco as fábricas de tintas e plásticos, que entre o término de uma tarefa e início de outra é necessário realizar a limpeza da máquina. Helal et al. (2006) implementaram uma Busca Tabu (Glover, 1997), utilizando perturbações intra-máquinas (perturbações entre as tarefas de uma máquina), e inter-máquinas (pertubações entre tarefas de máquinas diferentes). Além

[1] Instâncias e resultados disponíveis em `https://sites.wp.odu.edu/schedulingresearch/paper`

disso, os autores também utilizaram memória de longo prazo, baseada na posição em relação à tarefa anterior em cada máquina. Por outro lado, Arnaout et al. (2010) se basearam no Algoritmo de Colônia de Formigas (ACO) (Dorigo et al., 1996), dividindo-o em duas etapas. Na primeira etapa as tarefas são alocadas às máquinas e na segunda etapa são ordenadas. Em ambos os artigos, foram feitas comparações com utilizando as instâncias e os resultados disponibilizados online[1] além do algoritmo *Partitioning Heuristic* (PH), proposto por Al-Salem (2004). Arnaout et al. (2010), por ser um artigo mais recente, comparou seus resultados com Helal et al. (2006), também. Os resultados obtidos pelo ACO foram os melhores, comparado aos demais algoritmos comparados, em todas as instâncias.

Chang e Chen (2011), influenciados por fábricas e processos que podem ser modelados como UPMSP, propuseram a utilização da Propriedade de Dominância (*Dominance Properties* - DP). Nessa propriedade, realiza-se trocas entre tarefas na mesma máquina (inter-máquinas) e trocas entre máquinas diferentes (intra-máquinas), sendo verificado o valor de ajuste. Esse valor é a diferença entre o valor de solução anterior e o valor após a essa troca, sendo que quanto menor o valor de ajuste, melhor a solução. Para verificar a sua viabilidade, Chang e Chen (2011) implementaram as meta-heurística *Simulated Annealing* (SA) (Kirkpatrick et al., 1983) e Algoritmo Genético (*Genetic Algorithm* - GA) (Holland, 1975) utilizando o DP como método para a busca local.

Vallada e Ruiz (2011) constataram a escassez de estudos sobre o UPMSP e propuseram um GA para o problema. Para gerar a população inicial foi utilizado o algoritmo de Múltipla Inserção (Kurz e Askin, 2001) e para o mecanismo de seleção foram realizados torneios n-ários. Quando selecionados dois pais, é feito um cruzamento entre eles, gerando dois filhos. Porém, esse cruzamento retirava as tarefas das máquinas para inserir novamente ao problema. Dessa maneira, foi necessário realizar uma busca local para melhor inserção dessas tarefas. Vallada e Ruiz (2011) adaptaram a formulação de Guinet (1993) para o seu problema. Além disso, os autores implementaram quatro algoritmos genéticos, alterando somente em qual momento é realizada a busca local (durante o cruzamento e/ou após). Para avaliação desses algoritmos foram criadas instâncias com as seguintes características:

- Instâncias pequenas: com 6, 8, 10 e 12 tarefas e com 2, 3, 4 e 5 máquinas.
- Instâncias grandes: com 50, 100, 150, 200 e 250 tarefas e com 10, 15, 20, 25 e 30 máquinas.

Para cada combinação das características, foram criadas 10 instâncias, totalizando 640 instâncias pequenas e 1000 instâncias grandes, disponibilizadas online [2]. Vallada e Ruiz (2011) notaram que o GA com melhor desempenho foi aquele com as duas buscas locais implementadas. A partir desse algoritmo, foram implementados dois outros GAs, em que a diferença foi na calibração dos parâmetros.

[2]Instâncias disponíveis em `http://soa.iti.es` (ITI, 2019)

Com o objetivo de reduzir a diferença entre o progresso teórico e a prática das indústrias, Ying et al. (2012) propuseram a implementação da meta-heurística *Restricted Simulated Annealing* (RSA) para tratar o UPMSP. Utilizando os algoritmos propostos por Al-Salem (2004), Rabadi et al. (2006) e Helal et al. (2006), os autores comparam seus resultados baseados nas instâncias de Scheduling Research (2005). Para as instâncias pequenas, quase todas tiveram resultados muito próximos. Para as instâncias grandes, os autores obtiveram melhoras consideraveis, já que conseguiram superar os valores dos outros algoritmos.

Influenciados pelos trabalhos de Rabadi et al. (2006), Helal et al. (2006) e Arnaout et al. (2010), Fleszar et al. (2012) propuseram a implementação de um algoritmo *Multi-Start* (MS) com o *Variable Neighborhood Descent* (VND) (Hansen et al., 2001) para tratar do problema, nomeando-o de *Multi-Start VND* (MVND). Os movimentos utilizados para o VND foram de realocação de uma tarefa, troca de tarefas e realocação da sequência de uma parte, ou de todas tarefas, de uma máquina para outra. Os autores utilizaram as instâncias de Scheduling Research (2005) e os algoritmos propostos por Rabadi et al. (2006) (Meta-RaPS) e por Arnaout et al. (2010) (ACO), para avaliar o MVND. O MVND obteve resultados melhores do que o Meta-RaPS em $89,75\%$ dos casos, enquanto foi melhor em $97,16\%$ dos casos em relação ao ACO.

Pela escassez de métodos exatos para tratar o UPMSP, Tran e Beck (2012) utilizaram uma abordagem baseada em *Logic-Based Benders Decomposition* para resolver o problema. Os autores dividiram o problema em subproblemas com o objetivo de resolver a sequência em cada máquina, enquanto o problema mestre é responsável pela alocação de tarefas às máquinas. Os autores comparam sua decomposição com o modelo de Guinet (1993) adaptado, utilizando as instâncias pequenas de Scheduling Research (2005), limitando a no máximo três horas de execução. A decomposição requereu tempos inferiores aos do modelo para todos os casos. Em seguida, Tran e Beck (2012) realizaram uma relaxação na decomposição para obter tempos ainda melhores e comparam os resultados com Helal et al. (2006) e com Arnaout et al. (2010), utilizando as instâncias grandes de Scheduling Research (2005). Para as instâncias com menos de 100 tarefas, os autores obtiveram resultados piores do que o proposto por Arnaout et al. (2010), porém o tempo foi menor. Dessa maneira os autores notaram potencial para melhora, já que nestas últimas comparações, o algoritmo estava relaxado, podendo ter melhores resultados caso não estivesse relaxado.

Diana et al. (2013) adaptaram o algoritmo *Clonal Selection Algorithm* (CLONALG) (De Castro e Von Zuben, 2000) para o UPMSP, sendo possível dividi-lo em cinco fases. Na primeira fase tem-se a geração da solução inicial, na segunda tem-se a afinidade de cada elemento da população. Na terceira fase, são gerados clones da população sendo proporcionais à afinidade de cada elemento da população (quanto maior a afinidade, mais clones terão). Na quarta fase são realizadas mutações em alguns indivíduos e na última fase é realizada a seleção dos n melhores elementos. Para comparar, Diana et al. (2013) implementaram o GA2

de Vallada e Ruiz (2011), além de versões diferentes do CLONALG:

- CL1: A população inicial é gerada de forma aleatória e a afinidade é baseada no valor do *makespan* e a quanto tempo aquela solução foi gerada (quanto mais velha a solução, pior);
- CL2: A população inicial é gerada utilizando o GRASP (*Greedy Randomized Adaptive Search Procedure*) e a afinidade é feita utilizando somente o valor do *makespan*;
- CL3: A população inicial é gerada igual ao CL2 e a afinidade é feita igual ao CL1.

Foram utilizadas as instâncias grandes de Vallada e Ruiz (2011) para comparar os resultados. Entre os algoritmos propostos por Diana et al. (2013), o CL3 foi o que obteve os melhores valores, aperfeiçoando os resultados da literatura.

Avalos-Rosales et al. (2015) também notaram a escassez de modelos matemáticos para o problema e propuseram uma reformulação do mesmo, além de um algoritmo *Multi-Start* (MS) com o *Variable Neighborhood Descent* (VND). Os autores se basearam nos modelos de Rabadi et al. (2006), Vallada e Ruiz (2011) (muito similar ao primeiro) e Tran e Beck (2012) e criaram um novo conjunto de restrições, sendo um deles um novo cálculo para o *makespan*. A partir dos modelos de Rabadi et al. (2006) e Tran e Beck (2012), foram gerados outros quatro modelos, em dois deles são feitas substituições das restrições que calculam o *makespan*, e nos outros dois são adicionadas as restrições do cálculo do *makespan*. No algoritmo MS, a cada iteração é gerada uma nova solução e é realizado um refinamento em duas etapas, em que ambas utilizam o VND. Os VNDs implementados se diferenciam pelos movimentos: um caso realiza movimentos intra-máquina e outro realiza movimento inter-máquinas. O VND com movimentos inter-máquinas é aplicado na primeira etapa, enquanto o VND com movimento intra-máquinas é aplicado na segunda etapa. Em ambas as etapas, o algoritmo para quando não obter uma melhora. Foram feitos testes, primeiramente, sobre os seis modelos, utilizando as instâncias pequenas de Vallada e Ruiz (2011). O modelo de Tran e Beck (2012) que teve a adição das restrições do cálculo do *makespan*, foi aquele que obteve os melhores resultados com o menor tempo. Foram realizados mais testes sobre este modelo, utilizando as instâncias médias, sendo estas geradas pelos próprios autores, limitando a no máximo 3600 segundos o tempo de execução. As instâncias médias contêm 20, 30, 40, 50 e 60 tarefas e 2, 3, 4 e 5 máquinas, gerando um total de 600 instâncias. O modelo não foi capaz de resolver algumas dessas instâncias. Para os testes, Avalos-Rosales et al. (2015) implementaram três versões do MS:

- MAI: contém somente a primeira etapa
- MAII: contém somente a segunda etapa
- MAIII: contém as duas etapas

Além disso, foi implementado o GA de Vallada e Ruiz (2011). Para esses algoritmos, foram utilizadas as instâncias grandes de Vallada e Ruiz (2011). O MAIII obteve os melhores resultados em menor tempo.

Com o intuito de comparar métodos exatos com as heurísticas desenvolvidas para o UPMSP, Tran et al. (2016) concebeu duas decomposições, sendo utilizada a formulação de Avalos-Rosales et al. (2015). A primeira decomposição é uma adaptação de Tran e Beck (2012)(*Logic-Based Benders Decomposition* - LBBD), já que foi utilizada outra formulação. A outra decomposição é chamada de *Branch-and-check*, que é uma fusão dos procedimentos *Branch-and-bound* e LBBD. Foram utilizadas as instâncias pequenas de Scheduling Research (2005) para verificar a capacidade das duas decomposições, com limite de três horas de execução. O *Branch-and-check* encontrou soluções para todas as instâncias, enquanto o LBBD não foi capaz de resolver algumas das instâncias. Para as instâncias grandes, Tran et al. (2016) implementaram o LBBD com uma faixa de 2% de GAP de aceitação para resolver e comparam com valores obtidos por Rabadi et al. (2006), Helal et al. (2006) e Arnaout et al. (2010) disponíveis no site[1]. Nos resultados, os autores conseguiram melhores resultados em menor tempo, além de conseguir garantir soluções de qualidade, devido a utilizarem métodos exatos para resolvê-los.

Wang et al. (2016) propuseram a utilização de *Estimation of Distribuition Algorithm* (EDA) junto à heurística *Iterated Greddy* (IG), chamando-a de EDA-IG. EDA utiliza uma matriz de probabilidades do valor da solução do modelo. IG é composta por três fase: *i*) destruição, em que pode ser removida uma tarefa de uma máquina ou remover todas as tarefas de uma única máquina, *ii*) reconstrução, em que a(s) tarefa(s) removidas são inseridas em novas posições, *iii*) melhoria, em que é verificado se ocorreu melhora na solução. Essa heurística tem a capacidade de busca da EDA e a eficiência de algoritmo do IG. Para a análise de resultados foram implementados duas versões do EDA-IG, em que a diferença é que em uma é realizada a destruição de uma tarefa em uma máquina (EDA-IG1) e a na outra realizada a destruição de todas as tarefas na máquina (EDA-IG2). Para os testes, foram utilizadas todas as instâncias e os dois algoritmos genéticos propostos por Vallada e Ruiz (2011). Os resultados mostraram que o EDA-IG2 foi o algoritmo mais eficiente.

Santos et al. (2016b) notaram que cada vez mais, heurísticas e meta-heurísticas são geradas e criadas. Algumas utilizam de um abordagem mais simples, enquanto existem outras que utilizam outras heurísticas e meta-heurísticas para auxiliar na resolução de um problema. Porém, este último tipo de heurística e/ou meta-heurísticas contêm várias dificuldades para implementar, sendo as principais:

- Necessidade de muitos parâmetros;

- Exigência de muito tempo computacional, por conter várias fases para otimizar;

- Dificuldade na identificação da parte do algoritmo que tem maior eficiência no tratamento o problema;

- Necessidade de várias linhas de código para implementar, aumentando as chances de ocorrer *bugs* no código, sendo explicado de forma estatística em McConnell (2004).

Indo contra este tipo de algoritmo, os autores propuseram a implementação de algoritmos mais simples. As heurísticas implementadas foram SA, ILS (Lourenço et al., 2003), *Late Acceptance Hill-Climbing Heuristic* (LAHC), proposto por Burke e Bykov (2012) e uma heurística baseada em LAHC, chamada *Step Counting Hill-Climbing* (SCHS), introduzida por Bykov e Petrovic (2013). Os movimentos implementados foram realocação e troca em uma única máquina ou em diferentes máquinas, além de duas realocações ao mesmo tempo. Para análise dos dados, foram utilizadas as instâncias online[2] e implementado o algoritmo GA de Vallada e Ruiz (2011) para a comparação dos resultados. Além disso, foram feitas 5 repetições de cada algoritmo para cada uma das 1640 instâncias (640 pequenas e 1000 grandes). Sobre as instâncias pequenas, AIRP e LAHC produziram os melhores resultados. Das instâncias grandes, os algoritmos propostos por Santos et al. (2016b) obtiveram 901 valores melhores do que Vallada e Ruiz (2011), sendo que o SA obteve os melhores resultados.

Motivados por pesquisas que utilizam o *Adaptive Large Neighborhood Search* (ALNS) (Ropke e Pisinger, 2006), Cota et al. (2017) propuseram a utilização do ALNS incorporado ao *Learning Automata* (LA), intitulando de LA-ALNS. Além disso, os autores utilizam o "Método Húngaro", sendo este baseado no Algoritmo Húngaro, para resolver os subproblemas de alocação de tarefas em uma máquina já determinada. Para comparar resultados, foram utilizadas as instâncias de Rabadi et al. (2006), além dos algoritmos AIRP (Cota et al., 2014), ACO (Arnaout et al., 2010) e ACO II (Arnaout et al., 2014). Essas instâncias têm características de serem balanceadas (os tempos de processamento e de preparo com valores próximos), domínio no processamento (em que os tempos de processamento impactam mais nas máquinas) e domínio do preparo (em que os tempos de preparo impactam mais nas máquinas). Os autores conseguiram superar os outros algoritmos em 88% dos casos. Para comparação, os autores criaram instâncias com 200, 400, 600, 800 e 1000 tarefas e com 2, 4, 6 e 8 máquinas.

Fanjul-Peyro et al. (2019) propuseram um algoritmo de decomposição ao UPMSP, utilizando o modelo de Avalos-Rosales et al. (2015) e a decomposição de Tran et al. (2016). O modelo principal restringem a alocação das tarefas, enquanto que os modelos secundários são modelos para resolver o subproblema de ordenação de tarefas na máquina. Como os autores acreditavam que as instâncias não estão de acordo com problemas reais, eles criaram instâncias do problema, sendo as instâncias pequenas com 10, 20, 30 e 40 tarefas, instâncias médias com 40, 60, 80 ou 120 tarefas e instâncias grandes com 200, 400, 600, 800 e 1000 tarefas. Para todas as instâncias são consideradas 2, 4, 6 e 8 máquinas, sendo esta uma quantidade relativamente baixa, para o número de tarefas. Para fins comparativos, Fanjul-Peyro et al. (2019) implementaram a decomposição de Tran et al. (2016) e a heurística proposta por Arnaout et al.

(2010). Entre as instâncias pequenas, os resultados não foram muito destoantes, nem com relação à qualidade nem com relação ao tempo. Nas instâncias médias, somente Tran et al. (2016) obteve soluções ótimas, porém utilizaram todo tempo necessário. Já para instâncias grandes, o algoritmo proposto por Fanjul-Peyro et al. (2019) somente obteve soluções com 200 tarefas, com 2, 4 e 6 máquinas e 400 tarefas, com 2 e 4 máquinas. Para as outras instâncias grandes não se obteve resultados no tempo limite.

2.3.2 *Fix-and-Optimize*

Santos et al. (2016a) propuseram a utilização do método de fixação de parte do problema para otimizar um subproblema do Problema de Escalonamento de Enfermeiras. Neste problema é tratado o escalonado todas as enfermeiras em uma quantidade de dias, porém cada enfermeira tem um contrato que restringe os dias nos quais pode trabalhar, quantos dias necessita de descanso, entre outros elementos. Por utilizar o modelo matemático do problema, os autores fixaram alguns dias e troca de enfermeiras de modo a obter um modelo menor e mais simples (um submodelo), para assim poder resolve-lo de forma mais rápida. Para comparação, os autores utilizaram instâncias da International Nurse Rostering Competition[3]. Estas instâncias são dividas em *Long*, *Medium* e *Sprint*, sendo subdivididas de *Early*, *Hidden* e *Late*. Para as instâncias com características de *Long* ou *Medium* contém 5 instâncias para cada subdivisão delas. Já para as instâncias com característica *Sprint* contém 10 instâncias para cada subdivisão, totalizando 60 instâncias. Além disso, as instâncias na subdivisão de *Hidden* são consideradas instâncias difíceis de se trabalhar. No mesmo local onde contém as instâncias, estão disponíveis os melhores resultados obtidos pela literatura. Santos et al. (2016a) obtiveram alguns resultados iguais a literatura, e no restante foi superado, mostrando grande potencial na utilização dessa abordagem.

O Problema de Programação de Múltiplos Projetos com Restrição de Recursos Multimodo é um problema de determinar a ordem dos projetos baseado na sua precedência, já que um projeto pode depender de um ou mais projetos e pode bloquear outros. Toffolo et al. (2016) se propuseram a gerar um novo algoritmo que fosse capaz de gerar soluções rápida, utilizando o modelo matemático gerado para o problema. Como utilizar o modelo completo é muito complexo e demanda muito tempo, os autores utilizaram da fixação de parte do modelo, afim de reduzir o tamanho dele, assim tendo este modelo se torna mais simples para obter resultados, além de demandar menos tempo de processamento. Para resolução do problema, os autores determinam algumas janelas de tempo do problema para trabalhar, assim, quando obtém uma otimização nestas janelas de tempo, os tempos dos projetos serão reduzidos. Foram utilizados as instâncias e resultados do Desafio MISTA 2013 (Wauters et al., 2016) para comparar o algoritmo proposto com os resultados do Desafio MISTA. Nos resultados, Toffolo et al. (2016) obteve resultados 8% melhores para algumas instâncias. Porém para instâncias grandes, o

[3]http://www.kuleuven-kulak.be/nrpcompetition

algoritmo demorou a obter resultados, por isso não obtiveram resultados melhores que o MISTA. Ao final do desafio os autores alcançarem o terceiro lugar, sendo o único a utilizar o modelo para execução do projeto.

Holm et al. (2019) trabalhando no Problema de Programação de Cursos Universitários da ITC2019 (*International Timetabling Competition 2019*), também aplica o método de *Fix-and-Optimze* para resolver o problema. Este problema consiste em alocar horários de aula em salas de aula para cada cursos de modo que os estudantes possam ser atendidos em suas requisições de matrícula. Estes cursos incidem restrições fortes e fracas sobre a distribuição das aulas e alocação de salas. Holm et al. (2019) realizou um pré-processamento do modelo matemático, para retirar alguns horários e salas que nunca iriam ser satisfeitos. Além disso, reduziu o número restrições redundantes e dominantes para obter um modelo mais simples. O método utilizado pelos autores é fixação dos valores das variáveis que definem qual o horário e sala do curso e otimizando o submodelo gerado a partir dessas fixações de valores. Dessa maneira, o modelo se torna ainda mais simples para ser resolvido. Os autores alcançaram o primeiro lugar no ITC2019, obtendo melhores resultados em 24 das 30 instâncias do desafio.

2.3.3 Conclusão

Em suma, a maioria dos trabalhos relacionados tratam o UPMSP utilizando heurísticas, meta-heurísticas ou métodos exatos. Os métodos exatos resolvem as instâncias pequenas de forma rápida, independente das características das instâncias (ter muita tarefa, ter pouca máquina, ter relação tarefa e máquina alta), porém isso não se aplica às instâncias grandes. Por outro lado, existe trabalhos que utilizam a fixação de parte do problema e otimizar a outra parte, e todos estes obtiveram grandes progressos em suas áreas, mas não existe nenhum estudo desse método em UPMSP. Dessa maneira, este trabalho propõe uma hibridização dos métodos exatos e heurísticos, além desta proposta torna-se vanguardista com relação ao UPMSP.

Capítulo 3

Formulações

Este capítulo apresenta e avalia três formulações diferentes para o UPMSP, propostas por Rabadi et al. (2006), Vallada e Ruiz (2011) e Avalos-Rosales et al. (2015). Além disso, também são apresentadas as formulações e algoritmos propostos por Fanjul-Peyro et al. (2019).

A notação utilizada pelas formulações para modelar o UPMSP são apresentadas pelas Tabelas 3.1 e 3.2. A Tabela 3.1 apresenta os dados de entrada para o UPMSP enquanto a Tabela 3.2 apresenta as variáveis comuns a todas as formulações, exceto as formulações propostas por Fanjul-Peyro et al. (2019).

Tabela 3.1: Descrição dos dados de entrada para as formulações.

Símbolo	Descrição
N	Conjunto de todas tarefas
N_0	Conjunto de todas tarefas incluindo a tarefa artificial
M	Conjunto de todas máquinas
$s_{i,j,k}$	Tempo de preparo entre finalizar a tarefa j e inicializar a tarefa k, na máquina i
$p_{i,j}$	Tempo de execução da tarefa j na máquina i
U	Limite superior válido para o valor do *makespan*[1]

Tabela 3.2: Descrição das variáveis comuns utilizadas nas formulações.

Símbolo	Descrição
C_{max}	*Makespan*
$x_{i,j,k}$	Variável binária que recebe o valor 1 caso a tarefa k seja inicializada em sequência da tarefa j na máquina i e 0 caso contrário

Todas as formulações consideradas têm como objetivo minimizar o *makespan*, representado

[1] Os valores considerados foram retirados de trabalhos da literatura

pela Equação (3.1).

$$\text{Minimizar} \quad C_{max} \tag{3.1}$$

3.1 Modelo de Rabadi et al. (2006)

A formulação proposta por Rabadi et al. (2006), referida por $\mathcal{F}_R$, é apresentada pelas Equações (3.2)–(3.6). Estas equações consideram variáveis adicionais, apresentadas a seguir:

- C_j: tempo de término da tarefa j.

O valor de C_{max} é definido pelas Restrições (3.2) em que C_{max} recebe o maior tempo de término de todas as tarefas. As Restrições (3.3) asseguram que todas as tarefas serão executadas e as Restrições (3.4) certificam que toda máquina utilizada irá ter tarefa artificial como primeira tarefa. As Restrições (3.5) asseguram que o fluxo das tarefas. As Restrições (3.6) impedem a criação de ciclos que não incluem a tarefa artificial. Por exemplo, um ciclo contendo as tarefas 1, 2 e 1 não é permitido por estas restrições. Por outro lado, um ciclo em que contém as tarefas 0, 1, 0 é permitido. Porém, antes de gerar as Restrições (3.6) é realizada uma verificação. Se o tempo de preparo para a nova tarefa mais o tempo de execução dessa nova tarefa for maior ou igual ao limite superior U, então não será adicionada essa restrição e a variável correspondente terá valor fixado em zero. É utilizado $2U$ na formulação, pois, por exemplo, caso a tarefa $x_{i,j,k}$ receba o valor 1, sendo que o a variável C_j já tenha recebido um valor próximo de U e o $s_{i,j,k}$ e p_j somados forem igual a U, causará uma inviabilidade. Mas, para o pior caso a soma de $s_{i,j,k}$ e p_j será igual a U e C_j só terá valores menores que U, sendo assim, os valores sempre serão menores que $2U$. As Restrições (3.7) garantem que a tarefa artificial deve ter tempo igual a 0. Por fim, as Restrições (3.8)–(3.10) especificam os domínios das variáveis.

$$C_{max} \geq C_j \qquad \forall j \in N \tag{3.2}$$

$$\sum_{i \in M} \sum_{\substack{j \in N_0: \\ k \neq j}} x_{i,j,k} = 1 \qquad \forall k \in N \tag{3.3}$$

$$\sum_{k \in N} x_{i,0,k} = 1 \qquad \forall i \in M \tag{3.4}$$

$$\sum_{\substack{k \in N_0: \\ k \neq j}} x_{i,j,k} = \sum_{\substack{h \in N_0: \\ h \neq j}} x_{i,h,j} \qquad \forall i \in M, j \in N \tag{3.5}$$

$$C_k - C_j + 2U \left(1 - \sum_{i \in M} x_{i,j,k}\right) \geq \sum_{i \in M} \left(s_{i,j,k} + p_{i,k}\right) x_{i,j,k} \qquad \forall j \in N_0, k \in N, j \neq k \tag{3.6}$$

$$C_0 = 0 \tag{3.7}$$

$$x_{i,j,k} \in \{0,1\} \qquad \forall i \in M, j \in N_0, k \in N_0, j \neq k \tag{3.8}$$

$$C_j \geq 0 \qquad \forall j \in N \tag{3.9}$$

$$C_{max} \geq 0 \tag{3.10}$$

3.2 Modelo de Vallada e Ruiz (2011)

A formulação proposta por Vallada e Ruiz (2011), referida por $\mathcal{F}_V$, é apresentada pelas Equações (3.11)–(3.17). Estas equações consideram variáveis adicionais, apresentadas a seguir:

- $C_{i,j}$: representa o tempo da máquina i para concluir a execução da tarefa j.

As Restrições (3.11)–(3.13) são iguais às Restrições (3.2), (3.7) e (3.3), respectivamente. As diferenças são que, em relação às Restrições (3.14), são utilizada uma desigualdade ($\leq$) ao invés de uma igualdade. As Restrições (3.15) são iguais às Restrições (3.6), porém são utilizadas as variáveis $C_{i,j}$. Além disso, é gerada uma restrição por variável $x_{i,j,k}$, enquanto nas Restrições (3.6) são criadas restrições para cada par de tarefas. As Restrições (3.16) asseguram que toda tarefa deve ser executada por exatamente uma máquina, e as Restrições (3.17) garantem que todas as tarefas irão ser executadas por completo. As Restrições (3.18)–(3.20) especificam os domínios das variáveis.

$$C_{max} \geq C_{i,j} \qquad \forall i \in M, j \in N \tag{3.11}$$

$$C_{i,0} = 0 \qquad \forall i \in M \tag{3.12}$$

$$\sum_{i \in M} \sum_{\substack{j \in N_0: \\ k \neq j}} x_{i,j,k} = 1 \qquad \forall k \in N \tag{3.13}$$

$$\sum_{k \in N} x_{i,0,k} \leq 1 \qquad \forall i \in M \tag{3.14}$$

$$C_{i,k} - C_{i,j} + 2U(1 - x_{i,j,k}) \geq s_{i,j,k} + p_{i,k} \qquad \forall i \in M, j \in N_0, k \in N, j \neq k \tag{3.15}$$

$$\sum_{\substack{h \in N_0: \\ h \neq k \\ h \neq j}} x_{i,h,j} \geq x_{i,j,k} \qquad \forall i \in M, j \in N, k \in N, j \neq k \tag{3.16}$$

$$\sum_{i \in M} \sum_{\substack{k \in N_0: \\ k \neq j}} x_{i,j,k} \leq 1 \qquad \forall j \in N \tag{3.17}$$

$$x_{i,j,k} \in \{0,1\} \qquad \forall i \in M, j \in N_0, k \in N_0, j \neq k \tag{3.18}$$

$$C_{i,j} \geq 0 \qquad \forall i \in M, j \in N_0 \tag{3.19}$$

$$C_{max} \geq 0 \tag{3.20}$$

3.3 Modelo de Avalos-Rosales et al. (2015)

Recentemente, Avalos-Rosales et al. (2015) propuseram a formulação $\mathcal{F}_A$ apresentada pelas Equações (3.21)–(3.28). $\mathcal{F}_A$ considera as mesmas variáveis da formulação $\mathcal{F}_R$, além das variáveis descritas a seguir:

- $y_{i,j}$: variável binária que recebe 1 se a tarefa j for executada na máquina i e 0 caso contrário.

As Restrições (3.21), (3.22), (3.23) são iguais às Restrições (3.11), (3.12) e (3.14), respectivamente. Já as Restrições (3.24) são iguais às Restrições (3.15), porém são utilizadas as variáveis C_j. As Restrições (3.25) fazem a mesma função das Restrições (3.21), porém, ao invés de utilizar as variáveis C_j, é utilizada a soma do tempo total da máquina i. Avalos-Rosales et al. (2015) mostraram que manter as duas restrições resulta em melhor desempenho por parte do *solver*. As Restrições (3.26) definem que toda tarefa deve pertencer a uma máquina. Finalmente, as Restrições (3.27) e (3.28) definem que toda tarefa deve ser inicializada e processada, se ela for alocada àquela máquina. As Restrições (3.29) a (3.32) dizem respeito ao domínio das variáveis.

$$C_{max} \geq C_j \qquad \forall j \in N \tag{3.21}$$

$$C_0 = 0 \tag{3.22}$$

$$\sum_{k \in N} x_{i,0,k} \leq 1 \qquad \forall i \in M \tag{3.23}$$

$$C_k - C_j + 2U(1 - x_{i,j,k}) \geq s_{i,j,k} + p_{i,j} \qquad \forall i \in M, j \in N_0, k \in N, j \neq k \tag{3.24}$$

$$\sum_{j \in N_0} \sum_{\substack{k \in N \\ k \neq j}} s_{i,j,k} x_{i,j,k} + \sum_{j \in N} p_{i,j} y_{i,j} \leq C_{max} \qquad \forall i \in M \tag{3.25}$$

$$\sum_{i \in M} y_{i,j} = 1 \qquad \forall j \in N \tag{3.26}$$

$$y_{i,j} = \sum_{\substack{k \in N_0: \\ j \neq k}} x_{i,j,k} \qquad \forall i \in M, j \in N \tag{3.27}$$

$$y_{i,k} = \sum_{\substack{j \in N_0: \\ j \neq k}} x_{i,j,k} \qquad \forall i \in M, k \in N \tag{3.28}$$

$$x_{i,j,k} \in \{0, 1\} \qquad \forall i \in M, j \in N_0, k \in N_0, j \neq k \tag{3.29}$$

$$y_{j,k} \in \{0, 1\} \qquad \forall i \in M, j \in N_0 \tag{3.30}$$

$$C_j \geq 0 \qquad \forall j \in N \tag{3.31}$$

$$C_{max} \geq 0 \tag{3.32}$$

3.4 Modelos e algoritmos propostos por Fanjul-Peyro et al. (2019)

Fanjul-Peyro et al. (2019) propõem utilizar a formulação de Avalos-Rosales et al. (2015) com a decomposição de Tran et al. (2016). Esta decomposição, baseada no método *branch-and-check*, é um algoritmo que necessita de duas formulações, sendo que uma formulação, com nome *master*, é a formulação que contém as restrições das alocações das tarefas nas máquinas, enquanto que a outra formulação, com nome *sequence*, contém as restrições para a ordem as tarefas em cada máquina. Este método é referido como $\mathcal{F}_F$.

Formulação *master*

A formulação *master*, apresentada pelas Equações (3.33)–(3.38), tem como objetivo minimizar o valor do *makespan* (Equação 3.1).

$$\sum_{k \in N} x_{i,0,k} \leq 1 \qquad \forall i \in M \tag{3.33}$$

$$\sum_{j \in N_0} \sum_{\substack{k \in N \\ k \neq j}} s_{i,j,k} x_{i,j,k} + \sum_{j \in N} p_{i,j} y_{i,j} \leq C_{max} \quad \forall i \in M \tag{3.34}$$

$$\sum_{i \in M} y_{i,j} = 1 \qquad \forall j \in N \tag{3.35}$$

$$y_{i,j} = \sum_{\substack{k \in N_0: \\ j \neq k}} x_{i,j,k} \qquad \forall i \in M, j \in N \tag{3.36}$$

$$y_{i,k} = \sum_{\substack{j \in N_0: \\ j \neq k}} x_{i,j,k} \qquad \forall i \in M, k \in N \tag{3.37}$$

$$CORTES \tag{3.38}$$

$$x_{i,j,k} \in [0,1] \qquad \forall i \in M, j \in N_0, k \in N_0, j \neq k \tag{3.39}$$

$$y_{j,k} \in \{0,1\} \qquad \forall i \in M, j \in N_0 \tag{3.40}$$

$$C_{max} \geq 0 \tag{3.41}$$

Nota-se que as Restrições (3.33)–(3.37) são exatamente iguais às Restrições (3.23), (3.25), (3.26), (3.27) e (3.28), respectivamente. As Restrições *CORTES* (3.38) são baseadas na Equação (3.42), sendo adicionadas a cada vez que o *solver* encontra a solução ótima para formulação *master*.

$$CORTE(h) : C_{max} \geq C^{hi*}_{max} - \sum_{j \in N^h_i} (1 - y_{i,j}) \Theta_{h,i,j} \tag{3.42}$$

A Equação (3.42) considera algumas constantes adicionais, apresentadas pela Tabela 3.3.

Tabela 3.3: Descrição das constantes presentes na Equação (3.42).

Símbolo	Descrição
C_{max}^{hi*}	O valor do tempo total da máquina i, na iteração h, em que os valores de $x_{i,j,k}$ não estejam relaxados.
N_i^h	O conjunto de tarefas na máquina i, na iteração h.
$\Theta_{h,i,j}$	Constante representa a Equação (3.43). Esta Equação é gerada a partir da soma do tempo de execução da tarefa j na máquina i ($p_{i,j}$) somado ao maior tempo de preparação ($s_{i,j,k}$) entre o conjunto de tarefas na máquina i ($max_{k \in N_i^h, k \neq j}\{s_{i,j,k}\}$).

$$\Theta_{h,i,j} = p_{i,j} + max_{k \in N_i^h, k \neq j}\{s_{i,j,k}\} \tag{3.43}$$

Estas restrições são geradas, pois o valor de $x_{i,j,k}$ é relaxado, ou seja, estas variáveis podem receber valores reais entre 0 e 1 (apresentando na Restrição (3.39)). Dessa maneira, uma solução ótima nesta formulação pode ser composta por algumas variáveis $x_{i,j,k}$ com valores diferentes de 0 e 1. Estas Restrições *CORTES* são geradas para garantir que, caso uma solução contenha novamente o mesmo conjunto de tarefas para a máquina i, não seja um conjunto relaxado, ou seja, nenhum $x_{i,j,k}$ que pertença a esta máquina e a este conjunto tenha valores diferentes de 0 e 1.

Formulação *sequence*

Na formulação *sequence* são adicionadas a constante N, o conjunto de constante $Y_{i,j}$ e o conjunto de variáveis u_j, que são apresentados na Tabela 3.4.

Tabela 3.4: Descrição das constantes e das variáveis adicionadas na formulação *sequence* de Fanjul-Peyro et al. (2019).

Símbolo	Descrição
N	Constante suficientemente grande para não permitir o ciclo dentro da máquina.
$Y_{i,j}$	Constante recebe o valor de $y_{i,j}$, quando gerado uma solução baseado na formulação *master*.
u_j	Variável real que recebe valor do número de tarefas processadas antes da tarefa j na máquina onde j está alocada.

Nesta formulação o objetivo é minimizar a soma do tempo total considerando todas as máquinas, apresentado pela Equação (3.44).

$$\text{Minimizar} \quad \sum_{i \in M} \sum_{j \in N_0} \sum_{\substack{k \in N \\ k \neq j}} s_{i,j,k} x_{i,j,k} + \sum_{i \in M} \sum_{j \in N} p_{i,j} Y_{i,j} \tag{3.44}$$

As Restrições dessa formulação são apresentadas pelas Equações (3.45)-(3.50). As Restrições (3.45) são iguais às Restrições (3.23). As Restrições (3.46) e (3.47) são similares às Restrições (3.27) e (3.47), porém trocam as variáveis $y_{i,j}$ pelas constantes $Y_{i,j}$. Dessa maneira, quando $Y_{i,j}$ tem o valor igual a 1, significa que a tarefa j esta na máquina i. Caso $Y_{i,j}$ tem valor igual a 0, então a tarefa j não esta na máquina 0. As Restrições (3.48) asseguram que caso j seja anterior a k na máquina i ($x_{i,j,k} = 1$), então $\sum_{i \in M} x_{i,j,k}$ é igual a 1. Dessa maneira, é possível reduzir a restrição para $u_k \geq u_j + 1$. Caso isso não seja verdade ($x_{i,j,k} = 0$), então $\sum_{i \in M} x_{i,j,k}$ é igual a 0, sendo possível reduzir a restrição para $u_j \leq \mathsf{N} - 1$. As Restrições (3.49) e (3.50) asseguram que não exista subciclos nas máquinas, baseado nas restrições de Miller Tucker Zemlin. Já as Restrições (3.51) e (3.52) restringem os valores das variáveis.

$$\sum_{k \in N} x_{i,0,k} \leq 1 \qquad \forall i \in M \tag{3.45}$$

$$Y_{i,j} = \sum_{\substack{k \in N_0: \\ j \neq k}} x_{i,j,k} \qquad \forall i \in M, j \in N \tag{3.46}$$

$$Y_{i,k} = \sum_{\substack{j \in N_0: \\ j \neq k}} x_{i,j,k} \qquad \forall i \in M, k \in N \tag{3.47}$$

$$u_j - u_k + \mathsf{N} \sum_{i \in M} x_{i,j,k} \leq \mathsf{N} - 1 \quad \forall j \in N, k \in N, j \neq k \tag{3.48}$$

$$u_j + \sum_{i \in M} x_{i,0,j} \geq 1 \qquad \forall j \in N \tag{3.49}$$

$$u_j + (\mathsf{N} - 1) \sum_{i \in M} x_{i,0,j} \leq \mathsf{N} - 1 \quad \forall i \in M, k \in N \tag{3.50}$$

$$x_{i,j,k} \in \{0, 1\} \qquad \forall i \in M, j \in N_0, k \in N_0, j \neq k \tag{3.51}$$

$$u_j \geq 0 \qquad \forall j \in K \tag{3.52}$$

Algoritmo proposto por Fanjul-Peyro et al. (2019)

O pseudocódigo do método proposto por Fanjul-Peyro et al. (2019) é apresentado no Algoritmo 1. Este algoritmo recebe como entrada o problema (P), que contém o conjunto de tarefas (N), o conjunto de máquinas (M), o tempo de execução da tarefa j na máquina i ($p_{i,j}$) e o tempo de preparação entre as tarefas j e k na máquina i ($s_{i,j,k}$). Além disso, é necessário o tempo máximo de execução (max_{time}). Ao início do programa, são atribuídos os valores de vazio ($\emptyset$), falso ($False$), infinito ($+\infty$), max_{time} e 0 para as variáveis S, $STOP$,

$best_{value}$, $timer$ e h, respectivamente (linhas 2-6). A partir de agora, o programa irá executar em ciclos até que chegue ao limite máximo de tempo ($timer > 0$) ou quando a variável $STOP$ receber o valor $True$ (linha 7). No início de cada ciclo, variável h é incrementada em 1 o seu valor (linha 8). Em seguida é realizado a verificação se h tem valor igual á 1 (linha 9). Caso seja, a solução *master* (S_m) recebe o resultado da execução do *solver* utilizando a formulação *master* ($\mathfrak{F}_m$) tendo os limites do *gap* em 2% ou menos ou alcançando o tempo máximo de 90% de max_{time}(linha 10). Caso h não tenha o valor 1, a solução *master* (S_m) recebe a execução do *solver* utilizando $\mathfrak{F}_m$ até que o *solver* encontre uma solução viável (linha 12). Com a S_m gerada, as constantes Y da formulação *sequence* ($\mathfrak{F}_s$), recebem os valores das variáveis y de S_m (linha 13). Em seguida é atualizado o tempo na variável *timer* (linha 14). Neste momento, é verificado se o *makespan* do S_m é melhor que o melhor resultado encontrado até o momento (linha 15). Caso seja, então é gerada a solução S_s executando o *solver* utilizando $\mathfrak{F}_s$ (linha 16). É verificado se o *makespan* de S_s é melhor já encontrado (linha 17). Caso seja, o novo valor é salvo em $best_{value}$ e a solução S_s é salvo em S (linhas 18 e 19). Se S_m é ótimo em $\mathfrak{F}_m$, então é gerado um uma nova restrição $CORTE(h)$ em $\mathfrak{F}_m$ (linhas 20 e 21). Caso o *makespan* do S_m não seja o melhor resultado encontrado até o momento, é feito a verificação se S_m é uma solução ótima em $\mathfrak{F}_m$ (linha 23). Caso seja verdadeiro, $STOP$ recebe o valor $True$ (linha 24). Em seguida é feito mais uma atualização do valor de *timer* (linha 25). Com o ciclo encerrando, o resultado é a solução S (linha 26).

Algoritmo 1: Algoritmo de Fanjul-Peyro et al. (2019)

Entrada: (i) Instância do problema P (N, M, p_{ik}, s_{ijk}); (ii) max_{time}
Saída: Solução (S)

início
  $S \leftarrow \emptyset$
  $STOP \leftarrow False$
  $best_{value} \leftarrow +\infty$
  $timer \leftarrow max_{time}$
  $h \leftarrow 0$
  enquanto $STOP = False$ *e* $timer > 0$ **faça**
    $h \leftarrow h + 1$
    se $h = 1$ **então**
      $S_m \leftarrow$ Solução gerada a partir da $\mathfrak{F}_m$ com limite do $gap \leq 2\%$ ou o tempo máximo de $0.9 \times max_{time}$
    senão
      $S_m \leftarrow$ Uma solução gerada a partir da $\mathfrak{F}_m$
    $Y_{\mathfrak{F}_s} \leftarrow y_{S_m}$
    Atualize $timer$
    se $C_{max}^{S_m} < best_{value}$ **então**
      $S_s \leftarrow$ Solução gerada a partir de $\mathfrak{F}_s$ com tempo máximo $timer$
      se $best_{value} > C_{max}^{S_s}$ **então**
        $best_{value} \leftarrow C_{max}^{S_s}$
        $S \leftarrow S_s$
      se S_m *é solução ótima em* $\mathfrak{F}_m$ **então**
        $\mathfrak{F}_m \leftarrow \mathfrak{F}_m \cup$ CORTE(h)
    senão
      se S_m *é solução ótima em* $\mathfrak{F}_m$ **então**
        $STOP \leftarrow True$
    Atualize $timer$
  retorna S

3.5 Experimentos computacionais dos modelos

As formulações foram implementadas na linguagem de programação Python, versão 3.6, utilizando a biblioteca Python-MIP versão 1.0.29 (Toffolo e Santos, 2019) e o resolvedor MIP Gurobi 8.0 (Gurobi Optimization, 2018) para resolver as formulações. O *solver* foi executado em um Intel® Xeon® CPU E5620 @ 2.40 GHz, com 16 CPUs, 47 GB de memória RAM e sistema operacional CentOS Linux. Além disso, foi utilizado uma *thread* para execução do *solver*.

As instâncias propostas por Vallada e Ruiz (2011) foram utilizadas e o tempo limite de 3600 segundos (uma hora) foi respeitado em cada experimento. Estas instâncias são divididas

em pequenas e grandes, sendo que as instâncias pequenas tem 6, 8, 10 e 12 tarefas, e podem ter 2, 3, 4 ou 5 máquinas. As instâncias grandes tem 50, 100, 150, 200 ou 250 tarefas, e podem ter 10, 15, 20, 25 ou 30 máquinas.

A Tabela 3.5 apresenta o número médio de variáveis e de restrições geradas para cada conjunto de instâncias, enquanto a Tabela 3.6 apresenta as médias de *gaps* obtidos e os tempo necessários por cada resolver formulação para cada conjunto de instâncias. Por fim, a Figura 3.1 apresenta os gráficos do resultado dos *gaps* obtidos, a esquerda, e os tempo utilizado, em segundos, para resolver cada formulação.

Tabela 3.5: Número médio de variáveis e restrições por cada modelo para cada conjunto de instância.

Instância		# Variáveis			# Restrições		
|N|	|M|	$\mathcal{F}_R$	$\mathcal{F}_V$	$\mathcal{F}_A$	$\mathcal{F}_R$	$\mathcal{F}_V$	$\mathcal{F}_A$
6	2	106,00	87,00	120,00	70,00	153,83	99,83
	3	155,00	130,00	176,00	77,00	190,80	112,80
	4	204,00	173,00	232,00	84,00	216,20	114,20
	5	253,00	216,00	288,00	91,00	245,63	119,63
8	2	172,00	147,00	190,00	108,00	274,00	170,00
	3	253,00	220,00	280,00	117,00	362,75	210,75
	4	334,00	293,00	370,00	126,00	404,03	204,03
	5	415,00	366,00	460,00	135,00	442,70	194,70
10	2	254,00	223,00	276,00	154,00	424,00	254,00
	3	375,00	334,00	408,00	165,00	589,03	339,03
	4	496,00	445,00	540,00	176,00	684,53	354,53
	5	617,00	556,00	672,00	187,00	737,88	327,88
12	2	352,00	315,00	378,00	208,00	603,68	351,68
	3	521,00	472,00	450,00	221,00	878,68	506,68
	4	690,00	629,00	742,00	234,00	1.027,03	535,03
	5	859,00	786,00	924,00	247,00	1.090,18	478,18
Média		378,50	377,00	413,50	150,00	520,31	273,31

Na Tabela 3.5 é possível notar que a formulação $\mathcal{F}_V$ é a formulação que utilizou menos variáveis, enquanto a formulação $\mathcal{F}_A$ foi a formulação que utilizou a maior quantidade. É visível nessa tabela que a formulação $\mathcal{F}_R$ contém a menor quantidade de restrições, sendo uma discrepância para qualquer outra formulação. A formulação $\mathcal{F}_V$ utilizou um número maior de restrições. Por fim, a formulação $\mathcal{F}_F$ não está presente nesta tabela por considerar subproblemas de variados tamanhos para a resolução da formulação.

Tabela 3.6: Número médio de GAP e tempo por cada modelo para cada conjunto de instância.

Instância		*gap* (%)				Tempo (segundos)			
|N|	|M|	$\mathcal{F}_R$	$\mathcal{F}_V$	$\mathcal{F}_A$	$\mathcal{F}_F$	$\mathcal{F}_R$	$\mathcal{F}_V$	$\mathcal{F}_A$	$\mathcal{F}_F$
6	2	**0,00**	**0,00**	**0,00**	**0,00**	0,23	0,28	**0,03**	0,66
	3	**0,00**	**0,00**	**0,00**	**0,00**	0,12	0,11	**0,04**	0,79
	4	**0,00**	**0,00**	**0,00**	**0,00**	0,07	0,03	**0,02**	0,77
	5	**0,00**	**0,00**	**0,00**	**0,00**	0,05	0,02	**0,01**	0,78
8	2	**0,00**	**0,00**	**0,00**	**0,00**	11,68	9,75	**0,10**	0,77
	3	**0,00**	**0,00**	**0,00**	**0,00**	2,00	1,74	**0,16**	0,94
	4	**0,00**	**0,00**	**0,00**	**0,00**	0,36	0,28	**0,09**	0,97
	5	**0,00**	**0,00**	**0,00**	**0,00**	0,12	0,07	**0,04**	0,96
10	2	**0,00**	**0,00**	**0,00**	**0,00**	485,53	773,66	**0,26**	0,95
	3	**0,00**	**0,00**	**0,00**	**0,00**	29,00	44,42	**0,38**	1,07
	4	**0,00**	**0,00**	**0,00**	**0,00**	5,97	7,16	**0,40**	1,24
	5	**0,00**	**0,00**	**0,00**	**0,00**	1,23	0,97	**0,20**	1,22
12	2	40,61	47,38	**0,00**	**0,00**	3.382,30	3.459,25	**0,42**	1,06
	3	**0,00**	**0,00**	**0,00**	**0,00**	1.226,63	1.746,88	**0,70**	1,24
	4	**0,00**	**0,00**	**0,00**	**0,00**	145,53	204,79	**1,12**	1,48
	5	**0,00**	**0,00**	**0,00**	**0,00**	3,83	10,91	**0,79**	1,60
Média		2,54	3,19	**0,00**	**0,00**	189,66	391,27	**0,29**	0,97

A Tabela 3.6 apresenta os valores médios de *gap* e do tempo de processamento, em segundos, gasto para resolver cada grupo instância para cada formulação. No cálculo do *gap* é utilizada a Equação (3.53), considerando os limites superior (*upper bound* - *UB*) e inferior (*lower bound* - *LB*) obtidos pelo *solver*.

$$gap = \frac{UB - LB}{UB} \times 100 \tag{3.53}$$

Todas as formulações resultaram em *gap* igual a zero para todas as instâncias de 6 a 10 tarefas e para as instâncias de 12 tarefas com 3, 4 e 5 máquinas. Porém, para as instâncias de 12 tarefas e 2 máquinas, as formulações $\mathcal{F}_V$ e $\mathcal{F}_R$ resultaram em *gaps* bastante elevados, enquanto para as formulações $\mathcal{F}_A$ e $\mathcal{F}_F$ obtiveram valores 0 de *gap*. Para as formulações $\mathcal{F}_A$ e $\mathcal{F}_F$ foram necessários de no máximo um e dois segundos, respectivamente, para resolver qualquer instância. Considerando as outras formulações ($\mathcal{F}_V$ e $\mathcal{F}_R$), para as instâncias com 6 e 8 tarefas com qualquer quantidade de máquina, as formulações necessitaram de no máximo doze segundos. Para as instâncias com 10 tarefas e 3, 4 e 5 máquinas obtiveram tempo menor que 800 segundos. Para as instâncias com 12 tarefas e 5 máquinas, foram obtidos resultados com o tempo menor que onze segundos. Comparando essas instâncias com as restantes, é possível notar a diferença grande entre os tempos, existindo médias que são muito próximas

ao valor máximo de execução de 3600 segundos. As formulações $\mathcal{F}_A$ e $\mathcal{F}_F$ obtiveram *gap* zero para todas as instâncias. A formulação $\mathcal{F}_A$ foi a que demandou menor tempo para todas as instâncias.

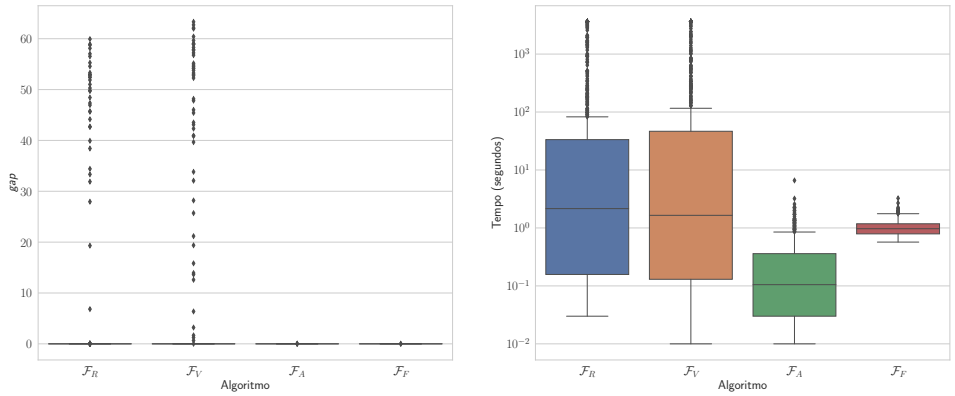


Figura 3.1: Gráfico dos resultados de *gaps* e tempos de processamentos das formulações.

A Figura 3.1 apresenta os gráficos dos *gaps* obtidos, a esquerda da figura, e do tempo de processamento, em segundos, a direita, gasto para resolver cada instância para cada formulação. No gráfico do tempo, as medidas estão em escalas logarítmicas. É possível notar, no gráfico dos *gaps*, que as formulações $\mathcal{F}_A$ e $\mathcal{F}_F$ não tem valores são superiores a 0, enquanto que as outras formulações têm seus valores variando entre 0 e 65. Além disso, no gráfico do tempo, as formulações $\mathcal{F}_A$ e $\mathcal{F}_F$ não ultrapassam a casa de 10 segundos, as outras formulações obtiveram valores superiores a 1.000.

Como as formulações $\mathcal{F}_A$ e $\mathcal{F}_F$ conseguiram encontrar soluções ótimas para todos os casos, estas formulações foram utilizadas para resolver as instâncias grandes de Vallada e Ruiz (2011), que contêm 50, 100, 150, 200 e 250 tarefas e com 10, 15, 20, 25 e 30 máquinas, totalizando 1.000 instâncias. Para cada combinação de quantidade de tarefas e máquinas tem-se 40 instâncias. O resultado dos experimentos com os modelos $\mathcal{F}_A$ e $\mathcal{F}_F$ estão apresentados nas tabelas 3.7 e3.8, respectivamente.

A Tabela 3.7 apresenta as médias do número de variáveis, restrições, *gap*, tempo, em segundos, e a quantidade de soluções encontradas para cada conjunto de instâncias no modelo $\mathcal{F}_A$. A quantidade de soluções ótimas (*gap* igual à zero) estão representadas entre parênteses para seus respectivos conjuntos de instâncias. Para quase todas as instâncias com 50 tarefas, a formulação obteve pelo menos 35 soluções das 40 instâncias. Porém, somente as instâncias com 30 máquinas obtiveram todos os valores ótimos. Para os conjuntos de instâncias com 100 tarefas, foram obtidas menos de 25 soluções por conjunto de instâncias. Além disso, somente três soluções foram obtidas, dentre todas as instâncias com 150 tarefas. Para o restante dos

Tabela 3.7: Número médio de variáveis, restrições, *gap*, tempo e quantidade de soluções para cada conjunto de 40 instâncias no modelo $\mathcal{F}_A$.

Instância		# Variáveis	# Restrições	*gap* (%)	Tempo (segundos)	# Soluções
|N|	|M|					
50	10	26.572	16.547,88	7,63	3.050,40	35(6)
	15	39.832	11.581,98	10,42	3.051,64	36(6)
	20	53.092	8.911,10	1,96	1.680,29	40(24)
	25	66.352	7.953,28	0,00	189,45	40(39)
	30	79.612	7.146,43	0,00	26,54	40(40)
100	10	103.122	101.807,60	8,19	3.600,66	23(0)
	15	154.632	105.314,13	7,15	3.600,55	4(0)
	20	206.142	81.498,18	12,40	3.600,60	12(0)
	25	257.652	64.511,48	13,55	3.600,45	10(0)
	30	309.162	54.020,15	19,10	3.601,22	13(0)
150	10	229.672	228.170,00	3,27	3.616,30	1(0)
	15	344.432	325.784,85	-	3.615,42	0(0)
	20	459.192	301.006,50	10,46	3.611,04	1(0)
	25	573.952	253.837,48	-	3.618,67	0(0)
	30	688.712	212.715,83	21,51	3.615,30	1(0)
200	10	406.222	404.220,00	-	3.627,02	0(0)
	15	609.232	606.230,00	-	3.600,61	0(0)
	20	812.242	701.403,03	-	3.600,31	0(0)
	25	1.015.252	645.762,38	-	3.603,71	0(0)
	30	1.218.262	551.123,78	-	3.604,31	0(0)
250	10	632.772	630.270,00	-	3.600,53	0(0)
	15	949.032	945.280,00	-	3.600,91	0(0)
	20	1.265.292	1.223.091,70	-	3.600,57	0(0)
	25	1.581.552	1.253.835,00	-	3.600,56	0(0)
	30	1.897.812	1.131.152,53	-	3.600,52	0(0)
Média		559.192,00	394.927,01	5,88	3.204,70	-
Total		-	-	-	-	256(115)

conjuntos de tarefas, não foram obtidas nenhuma solução. Portanto, mesmo utilizando a melhor formulação $\mathcal{F}_A$, a mesma encontrou 256 das 1000 instâncias grandes de Vallada e Ruiz (2011), encontrando somente 115 soluções ótimas.

A Tabela 3.8 apresenta as médias de *gap*, tempo, em segundos, e a quantidade de soluções para cada conjunto de instâncias no modelo $\mathcal{F}_A$. Para todas as instâncias, este modelo conseguiu alguma solução. Porém todas as soluções ótimas estão nos conjuntos de instâncias com 50 ou 100 tarefas, sendo que das 59 soluções ótimas, 57 estão no conjunto com 50 tarefas. Logo, este o modelo $\mathcal{F}_F$ conseguiu soluções para todos os casos, porém não conseguiu soluções

Tabela 3.8: Média do *gap*, do tempo e quantidade de soluções para cada conjunto de 40 instâncias no modelo $\mathcal{F}_F$.

Instância		*gap* (%)	Tempo (segundos)	# Soluções
|N|	|M|			
50	10	6,93	2605,34	40(6)
	15	13,53	2665,55	40(10)
	20	14,84	2584,24	40(11)
	25	13,34	2620,72	40(11)
	30	11,18	2198,36	40(19)
100	10	7,61	3510,77	40(0)
	15	18,25	3510,99	40(0)
	20	24,34	3428,97	40(1)
	25	29,26	3428,50	40(1)
	30	33,19	3511,64	40(0)
150	10	8,26	3511,74	40(0)
	15	15,88	3512,26	40(0)
	20	23,24	3512,85	40(0)
	25	29,08	3603,58	40(0)
	30	36,78	3515,26	40(0)
200	10	4,71	3590,17	40(0)
	15	11,93	3603,77	40(0)
	20	21,16	3608,06	40(0)
	25	30,55	3519,10	40(0)
	30	36,38	3614,27	40(0)
250	10	3,55	3603,65	40(0)
	15	12,44	3610,01	40(0)
	20	20,53	3616,88	40(0)
	25	33,08	3632,66	40(0)
	30	41,05	3634,92	40(0)
Média		20,04	3350,17	-
Total		-	-	1000(59)

ótimas somente para 59 das 1.000 instâncias.

Analisando ambas as Tabelas (3.7 e 3.8), é possível notar que $\mathcal{F}_A$ obteve mais soluções ótimas do que $\mathcal{F}_F$, porém $\mathcal{F}_F$ conseguiu soluções para todas as instâncias. As tabelas permitem concluir que mesmo as duas melhores formulações conhecidas até o momento para o UPMSP não foram capazes de resolver a maioria das instâncias consideradas.

Capítulo 4

Abordagem Proposta

Este capítulo apresenta o algoritmo híbrido proposto que combina programação matemática e busca local. O algoritmo proposto é composto por duas etapas: (i) etapa construtiva, responsável pela geração da solução inicial, e (ii) etapa de busca local, que utiliza um modelo matemático para resolver subproblemas e, desta forma, explorar vizinhanças grandes.

4.1 Algoritmo construtivo

O algoritmo construtivo inicialmente atribui as tarefas às máquinas. Para cada tarefa é selecionada a máquina que tem o menor tempo total. Caso exista mais de um máquina nesta situação é feito alocação da tarefa àquela máquina em que o tempo para processar esta tarefa seja o menor. Após a escolha da máquina, a tarefa é posicionado entre as demais desta máquina, de forma que o tempo de preparo entre as tarefas seja o mínimo.

O Algoritmo 2 apresenta como é gerado as soluções iniciais a serem utilizadas pela abordagem proposta. Este algoritmo recebe como entrada o conjunto de máquinas (M), o conjunto de tarefas (N - que não inclui a tarefa artificial 0), o tempo de execução da tarefa j na máquina i ($p_{i,j}$) e o tempo de preparo entre a finalização da tarefa j e início da tarefa k na máquina i ($s_{i,j,k}$). A saída do algoritmo é a solução S. Ao início do algoritmo, gera-se uma solução S vazia (linha 2) e em seguida as tarefas são embaralhadas (linha 3). Como a solução é gerada a partir da ordem de N, esse embaralhamento gera soluções diferentes, e potencialmente melhores, do que quando N está ordenado. A seguir, j recebe cada um dos valores em N (linha 4), sendo que no primeiro momento é realizada a escolha da máquina m na qual executará a tarefa j. Inicialmente m recebe o valor 0, referente à primeira máquina do conjunto máquinas (linha 5). A variável m representa a melhor máquina para alocar a tarefa j. A variável i recebe cada um dos valores de M e é realizado as seguintes comparações com m:

- Se o tempo total da máquina i for menor que da maquina m ($C_i < C_m$);

- Se o tempo total destas máquinas forem iguais ($C_m = C_i$) e o tempo de execução da tarefa j na máquina i for menor que de execução na máquina m ($p_{i,j} < p_{m,j}$).

Estas comparações estão apresentado na linha 7. Caso qualquer uma das comparações seja verdadeira, então significa que i é melhor candidato que m, dessa maneira m receber o valor de i (linha 8). Após a escolha da melhor máquina (m) a receber a tarefa j, é necessário definir em qual é a melhor posição para executar esta tarefa. A primeira posição possível para esta tarefa é a posição 0, como primeira tarefa da máquina m, dessa maneira ℓ recebe o valor 0 (linha 9). Na sequência, é feita uma comparação com cada posição q na máquina m para selecionar a melhor posição possível para a tarefa j (linha 10 a 12). Após a escolha da máquina m e da posição ℓ, é realizado a alocação da tarefa j na posição ℓ na máquina m (linha 13).

Algoritmo 2: Algoritmo Construtivo

Entrada: (i) Conjunto de máquinas (M); (ii) Conjunto de tarefas (N); (iii) tempo processamento da tarefa j na máquina i $(p_{i,j})$; (iv) tempo de preparação entre as tarefas j e k na máquina i $(s_{i,j,k})$.

Saída: Solução (S)

```
1  início
2      S ← ∅
3      Embaralhe N
4      para cada tarefa j ∈ N faça
           // Passo 1: Escolhendo uma máquina (m) para a tarefa
5          m ← 0
6          para cada máquina i ∈ M faça
7              se (C_i < C_m) ou (C_m = C_i e p_{i,j} < p_{m,j}) então
8                  m ← i
           // Passo 2: Posicionando a tarefa na máquina k
9          ℓ ← 0
10         para cada posição q na máquina m faça
11             se inserir tarefa j na posição q for mais barato que na posição ℓ então
12                 ℓ ← q
13         S ← S ∪ { alocação da tarefa j na posição ℓ da máquina m }
14     retorna S
```

4.2 Busca Local

A técnica chamada *Fix-and-Optimize* é uma técnica utilizada por alguns autores, como Santos et al. (2016a), Toffolo et al. (2016) e Holm et al. (2019). Esta técnica consiste em, a cada iteração do algoritmo, fixar valores de algumas variáveis do problema, dessa maneira o modelo irá se tornar mais simples de se resolver. Com o modelo mais simples, a resolução

deste modelo será mais rápida, mesmo que não seja possível encontrar a solução ótima para o modelo original nesta iteração. Então podemos resolver este modelo mais simples (submodelo) de forma a ter uma nova solução, melhor que a anterior, sem precisar trabalhar com todas as variáveis. Ou, no pior caso, já temos a melhor solução para este submodelo. De acordo com o resultado de cada iteração, o algoritmo poderá aumentar ou diminuir a quantidade de variáveis que terão valores fixos. Dessa maneira, podemos aumentar ou diminuir a complexidade do modelo de acordo com alguma condição.

O algoritmo proposto neste trabalho seleciona, a cada iteração, um subconjunto de tarefas a serem otimizadas. Em outras palavras, estamos fixando todos os elementos não foram selecionados e otimizado aqueles que foram selecionados, ou seja, *Fix-and-Optimize* (FixOpt). A partir do conjunto de tarefas é determinado também a quantidade de máquinas selecionadas (ou quantidade de máquinas livres). Máquinas e tarefas que não pertencerem a este conjunto são consideradas como máquinas e tarefas fixas. Máquinas fixas não podem receber ou perder tarefas de sua estrutura. Já tarefas fixas não podem ser removidas da máquina atual e somente realocados nesta máquina caso a realocação de uma outra tarefa reduza o tempo desta máquina. Caso exista duas ou mais tarefas fixas em sequência na mesma máquina, estas ficarão juntas e poderão ser realocadas em conjunto, ou seja, não pode realocar uma outra tarefa entre elas. Para cada iteração do processo tem-se as seguintes etapas:

- Seleção de tarefas e máquinas: Para selecionar as tarefas e máquinas livres, são realizadas escolhas de acordo com o tempo total de cada máquina na iteração atual. A máquina que o maior tempo atual (máquina *makespan*) sempre irá pertencer ao conjunto de máquinas livres, já que caso isso não ocorra, o *makespan* da solução não será afetado. O tamanho do conjunto de tarefas livres sempre tem que ser maior que quantidade de tarefas na máquina *makespan*, já que se pegarmos somente tarefas da *makespan*, seu valor não será alterado. A seleção de novas máquinas livres é baseada em roleta de probabilidade, em que cada máquina tem seu peso. Quanto maior o peso para a máquina i, maior as chances dessa máquina ser escolhida. O peso calculado para cada máquina é a diferença entre o tempo da máquina *makespan*e a máquina i, sendo que quanto maior a diferença de tempo, maior a chance desta máquina i ser escolhida. Dessa maneira, máquinas com menor tempo terão mais chances de ser selecionadas. Como estas máquinas têm o menor tempo, as chances de receberem uma tarefa da *makespan*e reduzirem o tempo da *makespan*é maior. Este peso é apresentado na Equação (4.1).

$$w_i = 1 - \frac{C_{max} - C_i}{C_{max}}, \forall i \in M \tag{4.1}$$

  Note que w_i é o peso para cada máquina i, sendo um valor entre 0 e 1. C_i é o tempo total da máquina i e C_{max} é o tempo total da máquina *makespan*. A escolha de novas máquinas livres cessa no momento em que a quantidade de tarefas no conjunto de

máquinas selecionadas é maior ou igual que a quantidade de tarefas destinadas a serem livres. Caso a quantidade de tarefas nestas máquinas seja maior que a quantidade de tarefas livres, são eleitas tarefas da última máquina selecionada até que se alcance a quantidade de tarefas livres. A última máquina escolhida, provavelmente, é a máquina com o maior tempo entre as demais escolhidas, assim, esta provavelmente será a que menos irá impactar nas alterações que podem ocorrer.

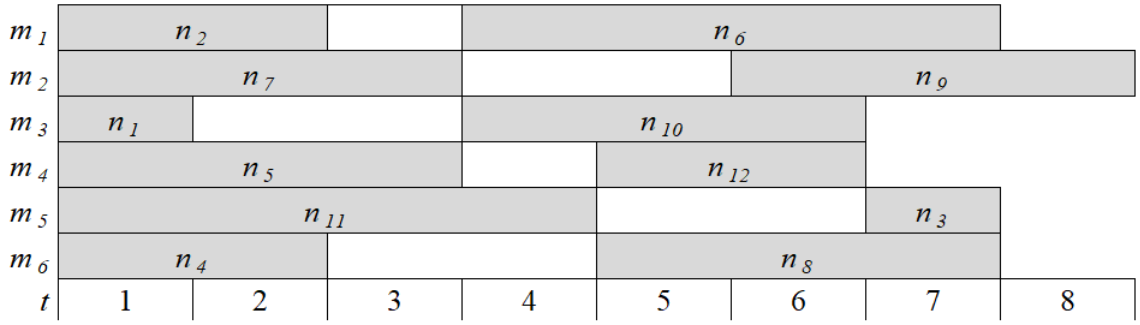
- Geração do submodelo: A partir do modelo original, são geradas todas as restrições sobre as máquinas livres e fixadas as tarefas que não pertencem ao conjunto de tarefas livres. Desta maneira, temos um modelo menor e mais simples para ser manipulado. Os modelos utilizados neste trabalho foram os modelos propostos por Avalos-Rosales et al. (2015) e por Fanjul-Peyro et al. (2019), conforme discutido no Capítulo 3.

- Geração da sub-solução: O valor inicial é gerado a partir da solução anterior, pegando somente os valores das máquinas e tarefas livres. É passado o submodelo e o valor inicial para a execução do *solver* que retorna uma sub-solução, utilizando no máximo uma quantidade de tempo definida como parâmetro.

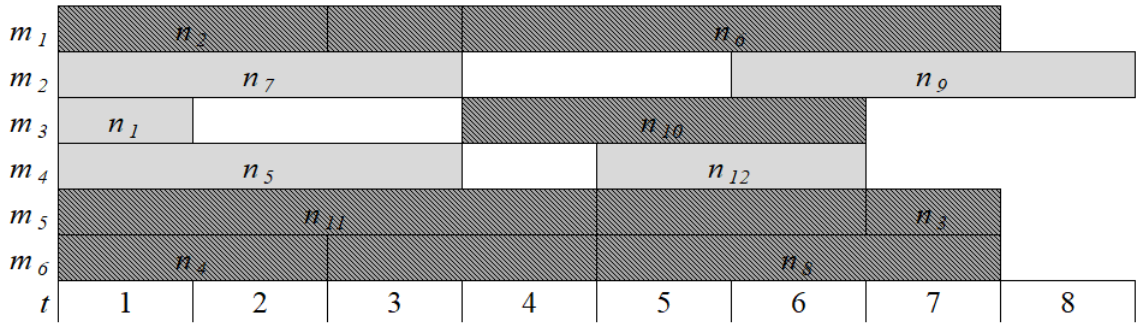
- Ajustes no número de tarefas livres: Como a execução do *solver* tem um limite de tempo, existe a chance de não ter encontrado a solução ótima. Caso não seja ótima, a quantidade de tarefas livres é reduzida em vinte porcento (20%). Caso o *solver* tenha encontrado a solução ótima, a quantidade de tarefas livres é ampliada para mais vinte porcento (20%) do valor atual. O aumento/redução de 20% do número de tarefas foi definido de forma empírica com intuito de criar novos subproblemas ligeiramente mais amplos/restritos que os anteriores. Caso fosse um valor muito baixo, quase não teria impacto no novo sub-modelo gerado. Caso fosse um valor muito alto, seria uma iteração em que o sub-modelo gerado seria muito grande para o tempo limite, e em outro seria muito rápido e com pouco impacto na solução.

A Figura 4.1 apresenta um exemplo de uma iteração do algoritmo. Nestas figuras, a primeira coluna representa as máquinas (m_1, m_2, m_3, m_4, m_5, m_6) e t é a linha do tempo. Em branco, sem nenhum conteúdo interno, são o tempo de preparação entre as tarefas. Os blocos com n_j representam o tempo de execução da tarefa, sendo j o número daquela tarefa. Neste exemplo, temos 6 máquinas e 12 tarefas e uma alocação atual de acordo com a Figura 4.1(a). Nesse exemplo considerou-se o número de tarefas livres $n = 5$. O conjunto de máquinas livre e tarefas livres são representados por $\mathcal{M}$ e $\mathcal{N}$, respectivamente. Como explicado anteriormente, a máquina *makespan* já pertence ao conjunto de máquinas livres e, por consequência, as tarefas que estão contidas nele também são livres. Neste caso, a máquina m_2 e as tarefas n_7 e n_9 são livres ($\mathcal{M} = \{m_2\}$ e $\mathcal{N} = \{n_7, n_9\}$). Como o conjunto de tarefas livres ainda é menor que o conjunto já selecionado de tarefas ($|\mathcal{N}| \leq n$), então é selecionada outra máquina. A nova máquina eleita foi a m_4, ou seja, uma nova máquina livre ($\mathcal{M} = \{m_2, m_4\}$). Como a

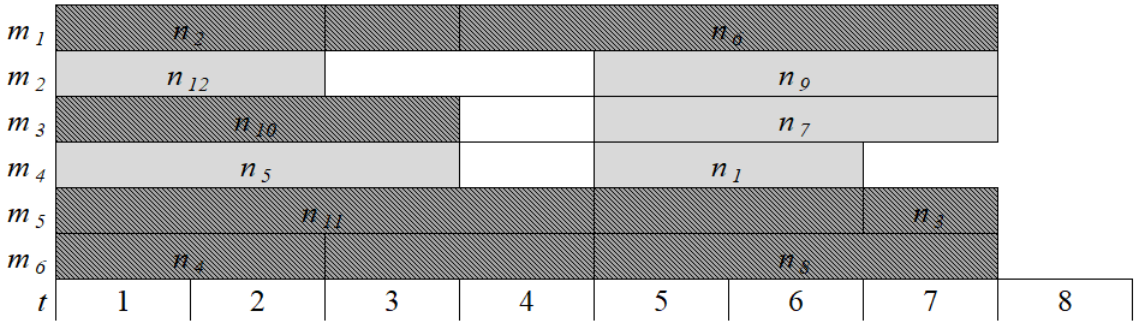
quantidade de tarefas em todas as máquinas selecionadas ($\mathcal{N}^{'} = \mathcal{N} \cup \{n_5, n_{12}\}$) ainda é menor que a quantidade de tarefas destinadas a serem livres ($|\mathcal{N}^{'}| \leq n$), então as tarefas n_5 e n_{12} são tarefas livres também ($\mathcal{N} = \{n_7, n_9, n_5, n_{12}\}$) e é selecionada outra máquina. A nova máquina eleita foi a m_3 ($\mathcal{M} = \{m_2, m_4, m_3\}$). Desta vez, a quantidade de tarefas em todas as máquinas selecionadas ($\mathcal{N}^{'} = \mathcal{N} \cup \{n_1, n_{10}\}$) é maior que a quantidade de tarefas destinadas a serem livres ($|\mathcal{N}^{'}| > n$). Então é eleita uma tarefa dentre as tarefas da máquina m_3 ($\{n_1, n_{10}\}$) para ser livre. A eleita foi a tarefa n_1, portanto a tarefa n_{10} está fixa à máquina m_3. Então ficamos com as máquinas m_2, m_4 e m_3 livres e as tarefas n_7, n_9, n_5, n_{12} e n_1 livres e este momento esta representado na Figura 4.1(b). Esta figura se diferencia da anterior por conter blocos mais escuros, sendo estes blocos representantes de tarefas ou máquinas fixas. Após a execução do *solver*, a partir do subproblema na Figura 4.1(b), foi gerada uma nova solução representada na Figura 4.1(c). Nesta nova solução, nota-se que as tarefas n_7, n_1 e n_{12} foram realocadas para as máquinas m_3, m_4 e m_2, respectivamente. Além disso, a tarefa n_{10} estava fixa na máquina m_3, porém ela foi reordenada para uma melhor solução.



(a) Solução atual S.



(b) Subproblema em que as máquinas m_1, m_5, m_6 e a n_{10} em m_3 estão fixas.



(c) Solução $S^{'}$ gerada a partir do subproblema na Figura 4.1(b).

Figura 4.1: Exemplo de uma iteração do FixOpt proposto.

4.3 Algoritmo

O pseudocódigo do algoritmo de busca local é apresentado pelo Algoritmo 3. O algoritmo recebe como entrada o problema (P), que contém o conjunto de tarefas (N), o conjunto de máquinas (M), o tempo de execução da tarefa j na máquina i ($p_{i,j}$) e o tempo de preparação entre as tarefas j e k na máquina i ($s_{i,j,k}$). Além disso, é necessário uma solução inicial S, o tempo de execução da heurística t_{max}, a quantidade de tarefas para o subproblema (n) e tempo limite de execução do subproblema (t). O algoritmo retorna uma nova solução refinada (S). Inicialmente é gerado um ciclo que executa toda a heurística até que alcance o tempo de máximo de t_{max} (linha 2). Neste ponto, a variável i^{max} recebe a máquina *makespan* da

solução S (linha 3). Esta máquina i^{max} é adicionada ao conjunto de máquinas livres $\mathcal{M}$ (linha 4). Além disso, as tarefas da máquina $S_{i^{max}}$ são adicionados ao conjunto de tarefas livres $\mathcal{N}$ (linha 5). Para garantir que serão selecionadas pelo menos duas máquinas, é selecionado o maior valor entre n a quantidade de tarefas na máquina $S_{i^{max}}$ mais um (linha 6). Neste momento irá adicionar tarefas em $\mathcal{N}$ até que alcance a quantidade de n tarefas (linha 7). No início desse ciclo, é selecionada uma máquina no conjunto de máquinas que não foram selecionadas ($M - \mathcal{M}$), utilizando a roleta com peso w_i como fator de escolha (linha 8). A máquina selecionada i é adiciona à $\mathcal{M}$ (linha 9). É feita a verificação se a quantidade em $\mathcal{N}$ mais a quantidade de tarefas nesta máquina (S_i) é menor ou igual a n (linha 10). Caso seja verdadeiro, então S_i é adicionado ao conjunto $\mathcal{N}$ (linha 11); caso contrário, são adicionadas tarefas aleatórias de S_i em $\mathcal{N}$ até que este conjunto tenha o tamanho n (linhas 13-15). Com os conjuntos $\mathcal{M}$ e $\mathcal{N}$ gerados, é gerado o modelo MIP ($\mathbb{M}$) utilizando estes conjuntos e os tempos execução e de preparação das tarefas nas máquinas (linha 16). Em $\mathbb{M}$ é inserido o valor da solução S (linha 17). Neste momento é executado o *solver* a partir de $\mathbb{M}$ tendo disponível no máximo t segundos de duração, que retorna uma solução nova S e *status* da solução (linha 18). A variável *status* pode retornar o resultado *Ótimo*, que representa S é uma solução ótima de $\mathbb{M}$. Caso este caso ocorra, o valor de n é incrementado em 20% do seu valor atual (linhas 19 e 20). Caso *status* não tenha sido *Ótimo*, é feito um decremento de 20% do valor atual de n (linha 22).

Algoritmo 3: Algoritmo Heurístico

Entrada: (i) Instância do problema P (N, M, p_{ik}, s_{ijk}); (ii) Solução Inicial S; (iii) Tempo total t_{max}; (iv) Quantidade de tarefas para o subproblema n; (v) Tempo limite do subproblema t.

Saída: Solução atualizada (S)

início
  enquanto Não ultrapassou o limite de t_{max} segundos **faça**
    // Passo 1: Escolhendo as tarefas e máquinas livres
    // Passo 1.1: *makespan* como máquina livre ($\mathcal{M}$)
    $i^{max} \leftarrow$ máquina *makespan* em S
    $\mathcal{M} \leftarrow \{i^{max}\}$
    // Passo 1.2: Adicionando as tarefas da *makespan* em $\mathcal{N}$
    $\mathcal{N} \leftarrow S_{i^{max}}$
    $n \leftarrow$ o maior valor entre n e $(|S_{i^{max}}| + 1)$
    enquanto $|\mathcal{N}| < n$ **faça**
      // Passo 1.3: Selecionando as outras tarefas e máquinas para serem livres
      $i \leftarrow$ máquina escolhida no conjunto $M - \mathcal{M}$ utilizando roleta com w_i
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{i\}$
      se $|\mathcal{N}| + |S_i| \leq n$ **então**
        $\mathcal{N} \leftarrow \mathcal{N} \cup S_i$
      senão
        enquanto $|\mathcal{N}| < n$ **faça**
          // Passo 1.4: Removendo as tarefas excedentes para que não seja maior que n
          $j \leftarrow$ selecione uma tarefa aleatória em S_i que não esteja em $\mathcal{N}$
          $\mathcal{N} \leftarrow \mathcal{N} \cup \{j\}$
    // Passo 2: Gerando submodelo
    $\mathbb{M} \leftarrow$ Modelo MIP gerado a partir do subproblema $\mathcal{P}(\mathcal{N}, \mathcal{M}, p_{ik}, s_{ijk})$
    Colocar solução S no modelo $\mathbb{M}$
    // Passo 3: Gerando nova solução
    $(S, status) \leftarrow$ Solução gerada a partir do modelo $\mathbb{M}$ no tempo máximo de t
    // Passo 4: Ajustes em n
    se *status* é Ótimo **então**
      $n \leftarrow n \times 1.2$
    senão
      $n \leftarrow n \times 0.8$
  retorna S

Capítulo 5

Resultados

Este capítulo apresenta e avalia os resultados das execuções do algoritmo proposto no Capítulo 4. Na Seção 5.1 é descrita a calibragem dos parâmetros do algoritmo proposto. Na Seção 5.2 são apresentados os resultados obtidos e uma breve comparação com o estado-da-arte da literatura. Na Seção 5.3 experimentos extensivos foram conduzidos e reportados sobre todas as instâncias grandes de Vallada e Ruiz (2011).

Para condução dos experimentos das Seções 5.1 e 5.3 foi utilizada uma máquina Intel® Xeon® CPU E5620 @ 2.40 GHz, com 16 CPUs, 47 GB de memória RAM e sistema operacional CentOS Linux. Já na Seção 5.2 foi utilizada uma máquina Dell G7-7588 A30P, com processador Intel Core i7-8750H (cache 9M, 2.20 GHz até 4.10 GHz), memória RAM de 16 GB com sistema Operacional Ubuntu 20.04 LTS. A linguagem de programação Python versão 3.6 com a biblioteca Python-Mip versão 1.0.29 Toffolo e Santos (2019) foi utilizada para implementar os algoritmos propostos. O software Gurobi versão 9.0 foi utilizado como resolvedor dos modelos matemáticos. Os experimentos das Seções 5.2 e 5.3 executaram com tempo limite de 3.600 segundos, enquanto que os experimentos da Seção 5.1 foram limitados de 1.800 segundos. Para todas as execuções utilizou-se uma única *thread*.

5.1 Calibragem de Parâmetros

Antes de conduzir experimentos sistemáticos sobre os algoritmos, foram conduzidos experimentos para determinar os melhores valores para os parâmetros de entrada do algoritmo: (i) Quantidade de tarefas para o subproblema (n); e (ii) Tempo limite de execução para o sub-problema (t). Os intervalos de valores para cada parâmetro calibrado são apresentados na Tabela 5.1.

O intervalo de 10 a 120 tarefas foi definido para o parâmetro n de forma empírica ao analisar a variação desse parâmetro ao longo da execução do algoritmo. Para t, o intervalo de 10 a 100 segundos foi considerado em consonância com os valores usualmente praticados na literatura Fonseca et al. (2016) Toffolo et al. (2016). As instâncias grandes contêm 50,

Tabela 5.1: Parâmetros, significado e intervalo de valores para calibragem do algoritmo.

Parâmetro	Significado	Intervalo de valores
n	Quantidade de tarefas iniciais para o subproblema	{10, 120} tarefas
t	Tempo limite de execução do sub-problema	{10, 100} segundos

100, 150, 200 ou 250 tarefas e 10, 15, 20, 25 ou 30 máquinas, totalizando 25 combinações de instâncias com diferentes dimensões. Note ainda que para cada dimensão foram propostas 40 instâncias, totalizando 1.000 instâncias grandes. A condução da calibragem dos parâmetros em todas as 1.000 instâncias disponíveis demandaria mais recursos computacionais e tempo que o disponível. Assim, optou-se por selecionar aleatoriamente uma instância de cada dimensão para compor o conjunto de arquivos para calibragem dos parâmetros. Ademais verificou-se que há pouca variação dentre as instâncias de uma mesma dimensão. Foram realizadas duas calibragem, em uma delas, o algoritmo proposto utiliza o modelo de Avalos-Rosales et al. (2015) e na outra utiliza o método de Fanjul-Peyro et al. (2019). Após a calibragem, os melhores valores para os parâmetros utilizando o modelo de Avalos-Rosales et al. (2015) foram 33 tarefas e 22 segundos de execução. Já os melhores valores utilizando o método de Fanjul-Peyro et al. (2019) foram 106 tarefas e 16 segundos de execução. Como todos os valores obtidos pela calibragem não estão nos extremos dos seus respectivos limites, foram considerados valores aceitáveis.

5.2 Resultados Obtidos

Considerando a inviabilidade em conduzir experimentos sobre todas as 1.000 instâncias disponíveis devido à limitação dos recursos disponíveis e à necessidade de replicação dos experimentos, foram selecionadas aleatoriamente outras 25 instâncias, uma de cada dimensão para avaliar os algoritmos propostos. De modo a permitir uma comparação dos resultados sob o mesmo ambiente computacional, foram implementados os métodos exatos de Avalos-Rosales et al. (2015) e de Fanjul-Peyro et al. (2019). O algoritmo proposto por Santos et al. (2016b), cujo código fonte foi disponibilizado online, também foi executado para reportar os resultados.

As instâncias e os resultados são apresentados na Tabela 5.2. A primeira coluna apresenta as instâncias abordadas. Para os resultados de Avalos-Rosales et al. (2015) (IP) e o método de Fanjul-Peyro et al. (2019) (MPA), são apresentadas as colunas *lower bound* (LB) e melhor valor encontrado pelo modelo/método (UB). Já os algoritmos de Santos et al. (2016b) (SLS), FixOpt utilizando o modelo de Avalos-Rosales et al. (2015) (IP FixOpt) e FixOpt utilizando o método de Fanjul-Peyro et al. (2019) (MPA FixOpt) foram executados 5 vezes para cada instância por conterem fatores de aleatoriedade. Dessa maneira, a tabela apresenta o melhor valor encontrado e a média de todos os resultados para cada instância. Além disso, estão ressaltados em negrito os melhores resultados dentre todos os métodos. Já os valores sublinhados são as

melhores soluções encontrado somente por, pelo menos, um dos algoritmos propostos. A comparação foi conduzida com esses métodos, pois são o estado da arte da literatura para o UPMSP com relação às instâncias de Vallada e Ruiz (2011). Para facilitar a leitura das tabelas e discussões as instâncias foram renomeadas de acordo com sua dimensão. A relação com os nomes originais é dada no Apêndice A.

Na Tabela 5.2 é possível notar que para as instâncias I_50_10 e I_150_10, os algoritmos propostos encontraram soluções melhores que a literatura. Já nas instâncias I_50_15 e I_50_20 ocorreram empates entre SLS e IP FixOpt. De forma similar, na instância I_100_15 ocorreu outro empate, porém entre SLS e MPA FixOpt. Para as instâncias I_100_20 e I_100_30, SLS e os algoritmos propostos obtiveram a mesma solução. Todos os algoritmos encontraram a solução ótima nas instâncias I_50_25 e I_50_30. Estas são soluções ótimas porque os métodos exatos (IP e MPA) obtiveram o valor de LB e Melhor iguais. O algoritmo MPA FixOpt obteve novas melhores soluções conhecidas para 7 das 25 instâncias. De forma similar, o SLS encontrou as melhores soluções dentre todos os algoritmos para 9 das 25 instâncias. Apesar dessa diferença, o algoritmo MPA FixOpt foi o que obteve a melhor média de melhores valores.

Se baseando no melhor algoritmo da literatura, o SLS, e no melhor algoritmo proposto, MPA FixOpt, foram gerados gráficos de convergência para cada instância executada nesta seção. Os gráficos de convergência apresentam o quão rápido os algoritmos encontraram o resultado final, sendo cada ponto neste gráfico apresenta o gap_{BKS} da nova solução e em qual tempo foi obtido esta solução. A Equação 5.1 apresenta o cálculo do gap_{BKS} de um determinado valor para o BKS.

$$gap_{BKS} = \frac{Valor - BKS}{Valor} \times 100 \tag{5.1}$$

As melhores soluções conhecidas (*Best Know Solutions* - BKS) para cada instância estão reportadas online[1], porém esses valores estão desatualizados (e defasados) e não consideram sequer as novas soluções dos algoritmos existentes MPA e SLS. Dessa forma, para os analisar adequadamente os resultados desses experimentos, foram consideradas como BKS a melhor solução obtida dentre os métodos estado da arte previamente existentes MPA e SLS. Os gráficos de convergência estão representados nas Figuras 5.1, 5.2, 5.3, 5.4 e 5.5, considerando as instâncias com 50, 100, 150, 200 e 250 tarefas, respectivamente. Cada figura contém cinco gráficos, que representam as convergências das instâncias com 10, 15, 20, 25 e 30 máquinas. Estes gráficos apresentam a relação entre tempo, em segundos, e gap_{BKS}, sendo o eixo do tempo esta em escala logarítmica. As linhas com x nos gráficos apresentam os resultados obtidos pelo MPA FixOpt, enquanto os gráficos com bola apresentam os resultados do SLS.

[1] Instâncias disponíveis em `http://soa.iti.es` (ITI, 2019)

Tabela 5.2: Comparação de resultados entre os algoritmos propostos (IP e MPA FixOpt) e o estado da arte da literatura (IP, MPA, and SLS).

Instâncias	IP (Avalos-Rosales et al. 2015)		MPA (Fanjul-Peyro et al. 2019)		SLS (Santos et al. 2016b)		IP Fix-Opt		MPA Fix-Opt	
	LB	Melhor	*LB*	Melhor	Melhor	Média	Melhor	Média	Melhor	Média
I_50_10	100	111	102	143	111	113,00	**110**	112,00	**110**	111,00
I_50_15	51	67	51	90	**58**	58,00	**58**	58,80	59	59,40
I_50_20	29	29	28	31	**29**	29,80	**29**	29,00	30	30,40
I_50_25	20	**20**	20	**20**	**20**	20,00	**20**	20,00	**20**	20,00
I_50_30	16	**16**	16	**16**	**16**	16,00	**16**	16,00	**16**	16,00
I_100_10	186	222	186	235	203	206,40	202	205,80	**200**	202,60
I_100_15	68	83	69	83	**76**	76,60	77	78,80	**76**	77,20
I_100_20	68	82	63	108	**74**	75,20	**74**	77,20	**74**	77,20
I_100_25	42	73	41	85	**52**	53,20	55	55,40	55	57,20
I_100_30	21	26	21	24	**23**	23,00	**23**	23,00	**23**	23,40
I_150_10	223	269	224	247	246	246,60	**235**	236,80	**235**	236,00
I_150_15	100	139	100	120	110	110,40	110	112,00	**109**	109,60
I_150_20	61	69	61	65	**63**	63,80	64	65,00	64	64,20
I_150_25	68	278	67	138	**82**	83,20	89	91,40	85	88,80
I_150_30	28	35	28	31	**30**	30,20	31	31,00	31	31,80
I_200_10	238	259	239	245	252	253,00	246	246,00	**243**	244,00
I_200_15	158	248	158	192	180	181,00	175	176,40	**173**	175,20
I_200_20	114	166	114	170	**134**	137,60	141	143,20	135	138,40
I_200_25	67	164	67	90	**76**	76,80	80	81,20	78	79,40
I_200_30	62	150	62	118	**76**	76,40	85	90,40	84	86,00
I_250_10	352	548	351	373	395	400,40	369	371,60	**365**	367,20
I_250_15	125	145	125	129	129	129,00	128	129,00	**127**	128,60
I_250_20	105	147	105	119	117	118,00	117	118,80	**115**	116,00
I_250_25	103	221	103	154	**124**	124,80	133	136,00	131	132,60
I_250_30	58	19452	58	79	**67**	67,60	72	73,20	71	71,80
Média	98,52	920,76	98,36	124,2	109,72	110,8	109,56	111,12	**108,36**	109,77

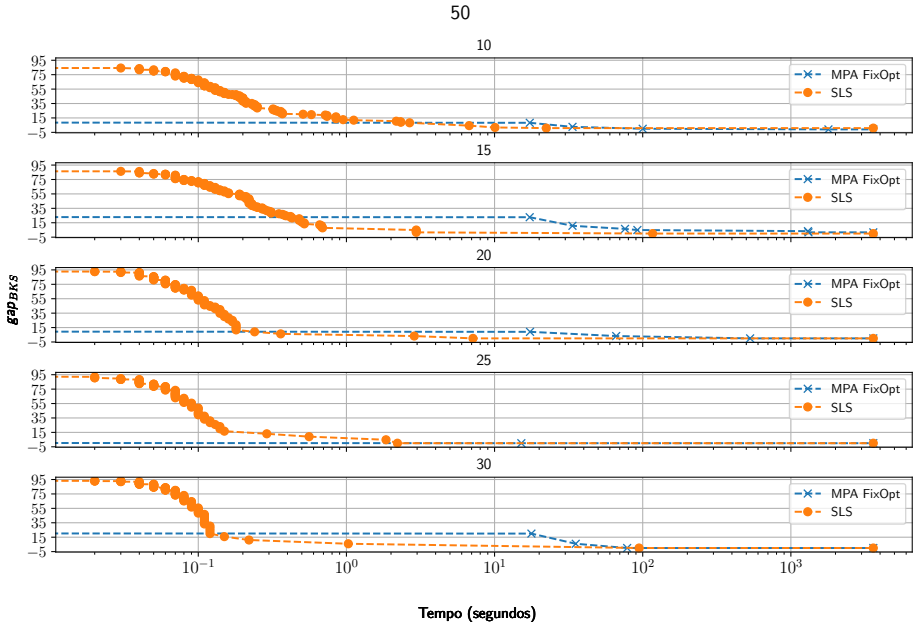


Figura 5.1: Gráficos de convergência do gap_{BKS} dos algoritmos SLS e MPA FixOpt para para o conjunto de instâncias com 50 tarefas.

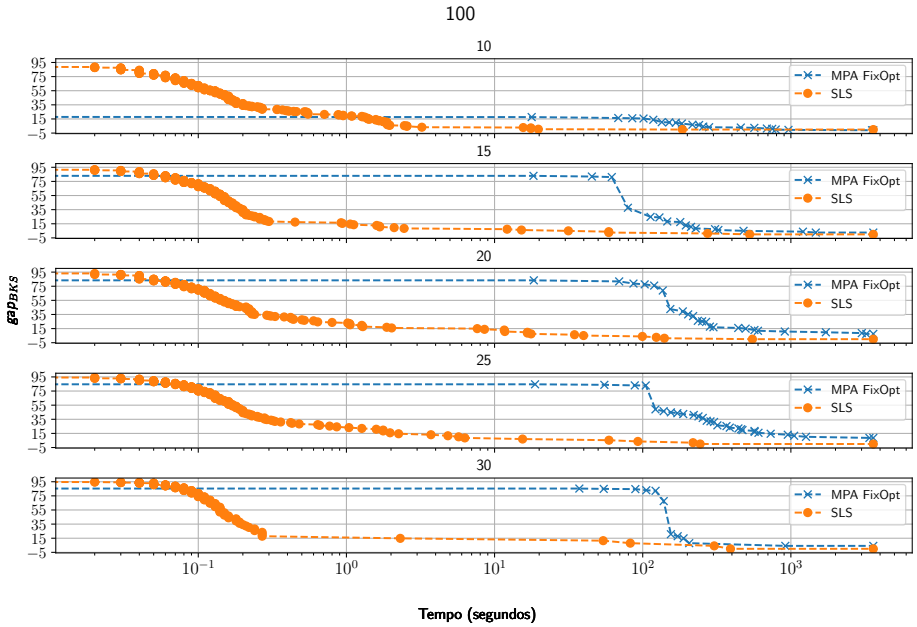


Figura 5.2: Gráficos de convergência do gap_{BKS} dos algoritmos SLS e MPA FixOpt para para o conjunto de instâncias com 100 tarefas.

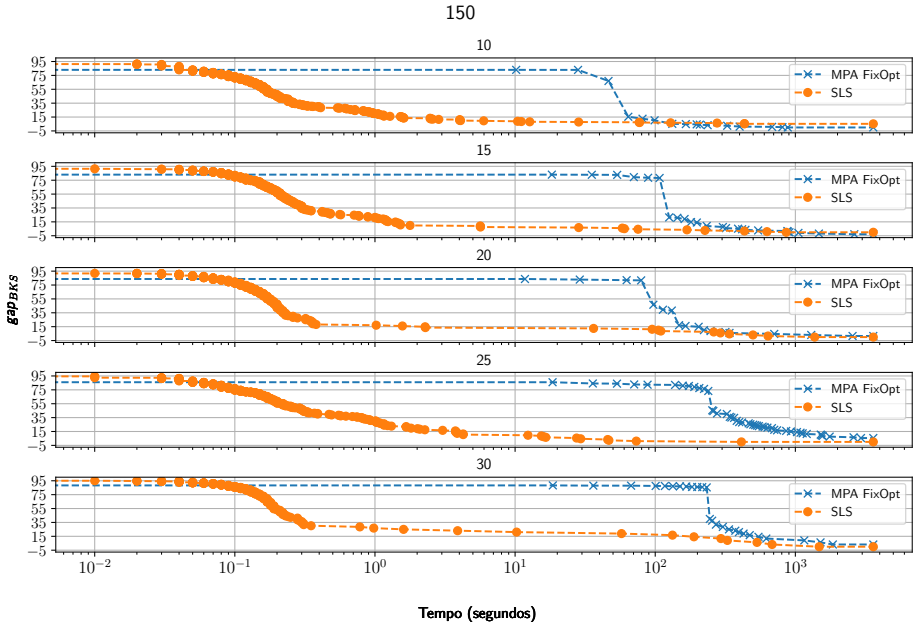


Figura 5.3: Gráficos de convergência do gap_{BKS} dos algoritmos SLS e MPA FixOpt para para o conjunto de instâncias com 150 tarefas.

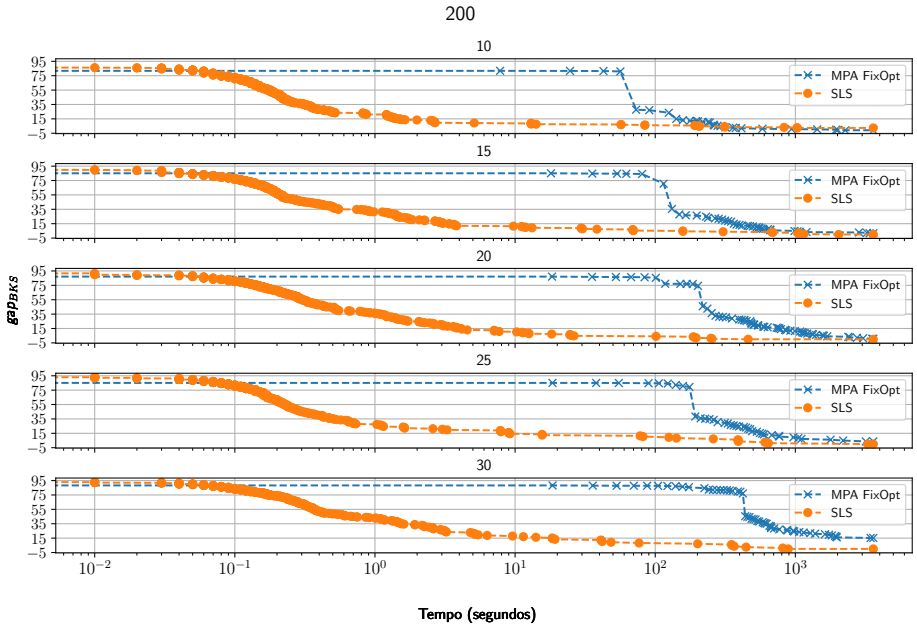


Figura 5.4: Gráficos de convergência do gap_{BKS} dos algoritmos SLS e MPA FixOpt para para o conjunto de instâncias com 200 tarefas.

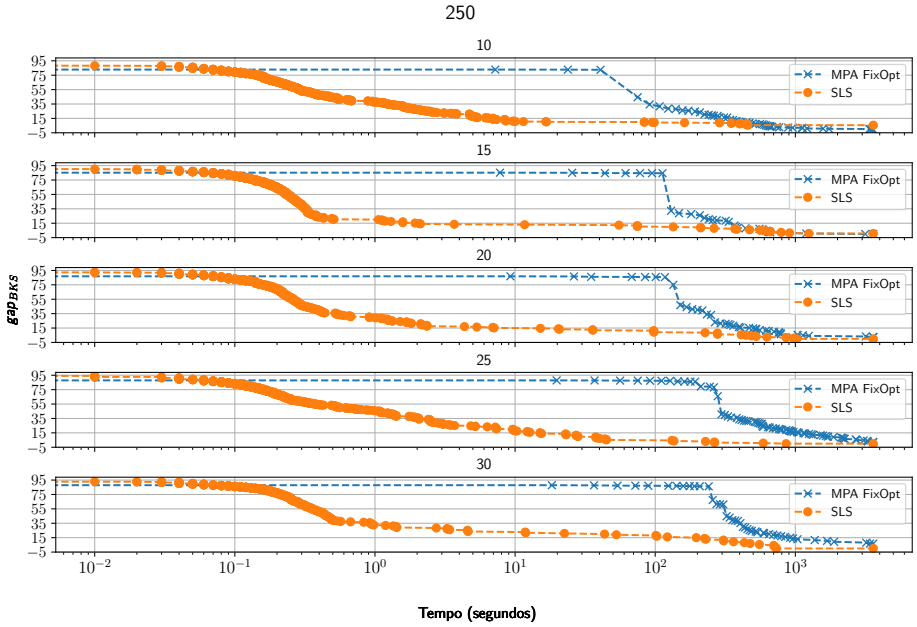


Figura 5.5: Gráficos de convergência do gap_{BKS} dos algoritmos SLS e MPA FixOpt para para o conjunto de instâncias com 250 tarefas.

Nas figuras 5.1 a 5.5 é possível notar que o MPA FixOpt obteve resultados inicias melhores que o SLS. Além disso, o SLS é o algoritmo que converge mais rápido, sendo que sua convergência ocorre antes dos 10 segundos, enquanto o MPA FixOpt obteve a convergência entre 100 e 1.000 segundos.

5.3 Experimentos com Todas as Instâncias

Na seção anterior foi possível concluir que os algoritmos de melhor desempenho no geral foram o SLS e o MPA FixOpt proposto. Nesta seção serão conduzidos experimentos computacionais mais aprofundados, considerando todas as 1.000 disponibilizadas por Vallada e Ruiz (2011). Para fins de comparação, esses experimentos também foram conduzidos para o melhor método exato para o UPMSP (MPA). Para esta execução, foi utilizada mesma máquina da Seção 5.1 com as mesma configurações da Seção 5.2. Mais uma vez, por conterem fatores de aleatoriedade e por falta de tempo para fazer mais repetições, os algoritmos SLS e MPA foram executados por apenas 5 vezes para cada instância. Já o MPA, por ser um algoritmo determinístico e exato, foi executado apenas uma vez por instância. Da mesma forma que na seção anterior, todos os algoritmos executaram até o tempo limite de 3.600 segundos (uma hora) em apenas uma *thread*.

A Figura 5.6 apresenta o gráfico dos $gaps_{BKS}$ obtidos por cada um dos algoritmos. Para construção desse gráfico foi considerada a melhor solução obtida para cada instância dentre

as 5 execuções de cada método.

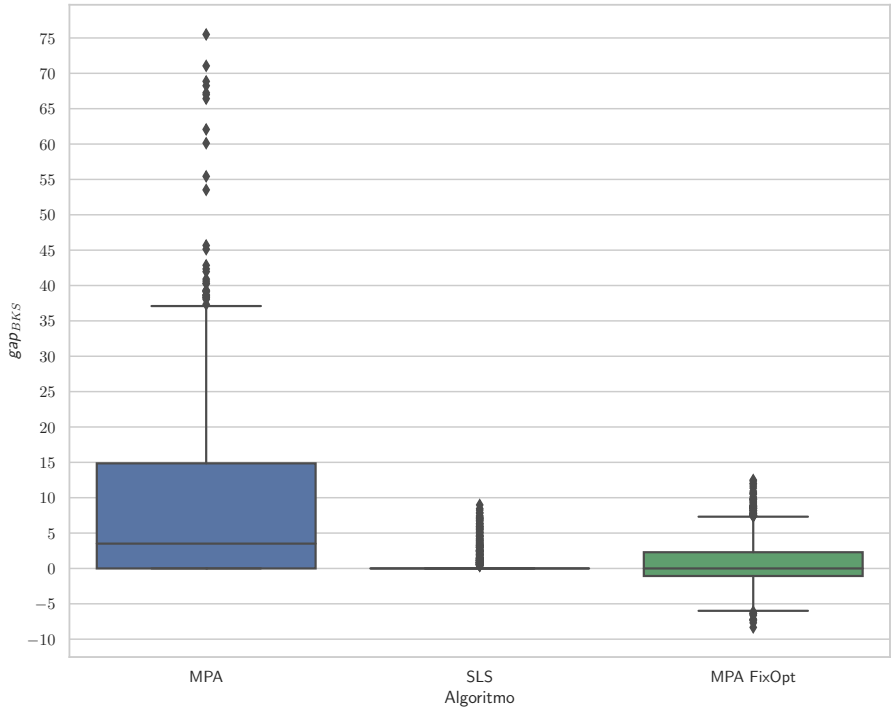


Figura 5.6: Gráfico dos $gaps_{BKS}$ de todas as instâncias para cada um dos métodos.

Na Figura 5.6 é possível notar que o SLS obteve gap 0 para a boa parte das instâncias. Já o MPA teve um gap maior que 15% para metade das instâncias. O algoritmo FixOpt proposto conseguiu um gap negativo (novo BKS) ou igual 0, para mais de 50% das instâncias.

Além da Figura 5.6, foram gerados outros 25 gráficos exibindo esses resultados agrupados pela dimensão das instâncias. Estes gráficos estão apresentados nas Figuras 5.7, 5.8, 5.9, 5.10 e 5.11. Cada figura representa as instâncias com 50, 100, 50, 100 e 250 tarefas, respectivamente. Em cada figura são apresentados 5 gráficos, sendo que cada um desdes gráficos representam o conjunto com 10, 15, 20, 25 e 30 máquinas. De mesma maneira que a Figura 5.6, cada gráfico apresenta o resultado de cada algoritmo e seu gap_{BKS}, considerando a melhor solução obtida dentre as 5 execuções.

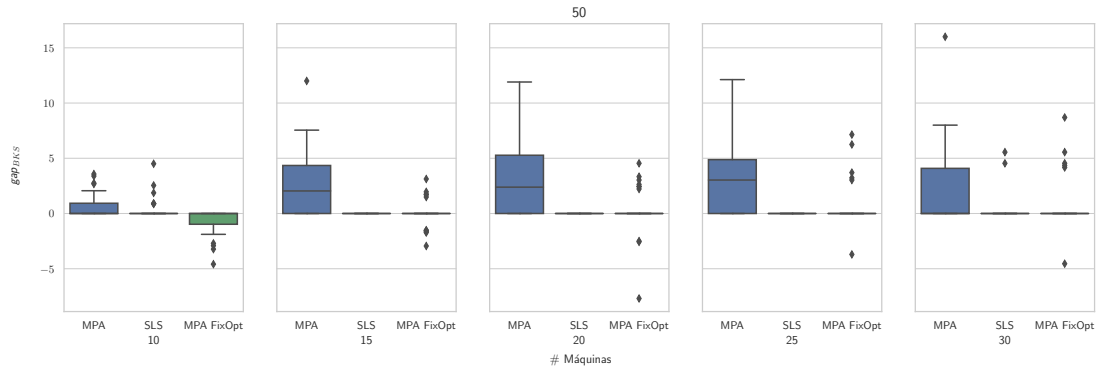


Figura 5.7: Gráfico dos gap_{BKS} obtidos pelos algoritmos MPA, SLS e MPA FixOPT para o conjunto de instância com 50 tarefas.

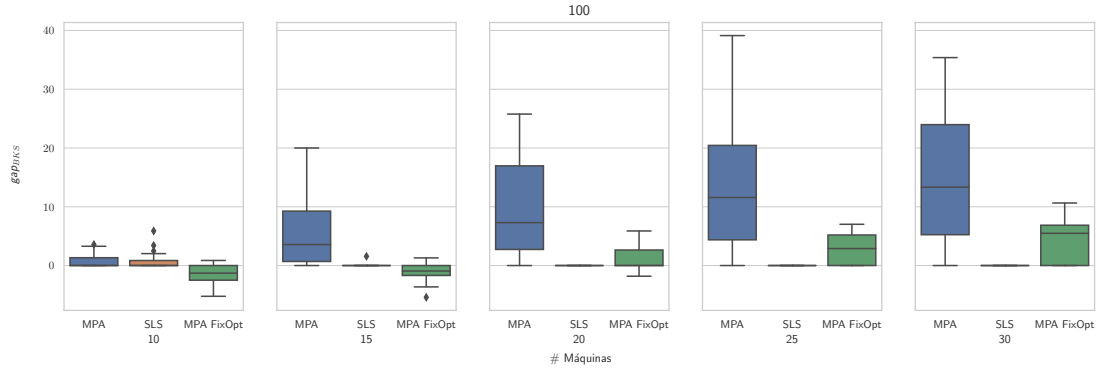


Figura 5.8: Gráfico dos gap_{BKS} obtidos pelos algoritmos MPA, SLS e MPA FixOPT para o conjunto de instância com 100 tarefas.

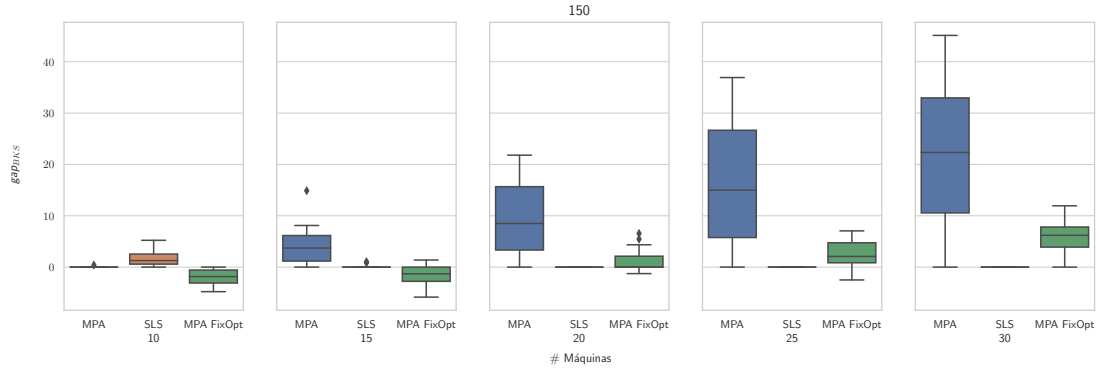


Figura 5.9: Gráfico dos gap_{BKS} obtidos pelos algoritmos MPA, SLS e MPA FixOPT para o conjunto de instância com 150 tarefas.

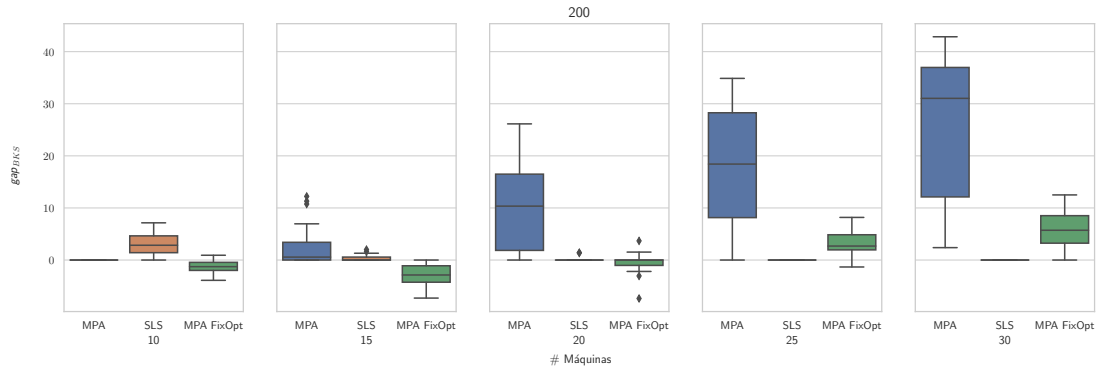


Figura 5.10: Gráfico dos gap_{BKS} obtidos pelos algoritmos MPA, SLS e MPA FixOPT para o conjunto de instância com 200 tarefas.

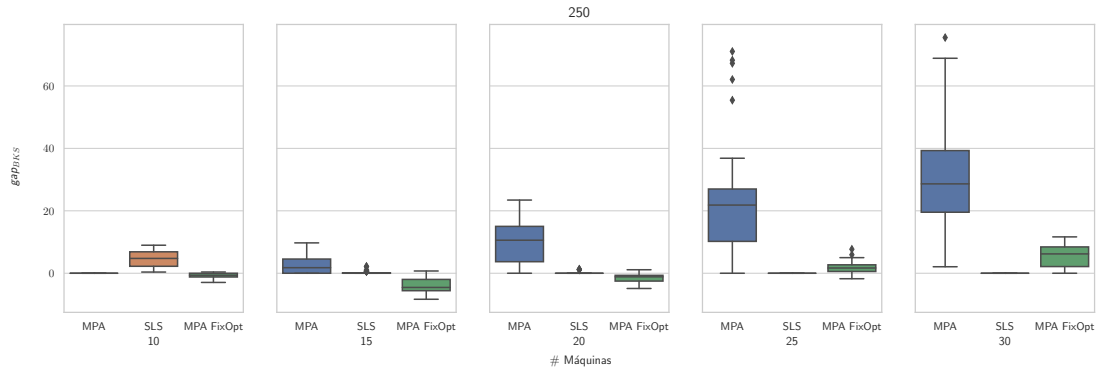


Figura 5.11: Gráfico dos gap_{BKS} obtidos pelos algoritmos MPA, SLS e MPA FixOPT para o conjunto de instância com 250 tarefas.

Nas Figuras 5.7, 5.8, 5.9, 5.10 e 5.11 podemos identificar que o algoritmo proposto teve um desempenho consideravelmente superior aos demais nas instâncias com um número menor de máquinas (10 e 15). Esse comportamento pode ser explicado pela natureza do método exato MPA, utilizado no método MPA FixOpt. O algoritmo MPA decompõe o problema em subproblemas que envolvem o sequenciamento das tarefas em cada máquina. Assim, quando menor o número de máquinas, mais difícil será cada subproblema e mais efetiva será a estratégia de decomposição prevista no MPA. Já para instâncias com um número maior de máquinas (25 e 30) a eficiência dos algoritmos metaheurísticos tradicionais (SLS) se sobressai.

A Tabela 5.3 apresenta a média do gap_{BKS}, RPD e quantidade de soluções obtidas para cada algoritmo, considerando as 40 instâncias de cada combinação de número de tarefas e máquinas. A seguir, é dado o significado de cada coluna desta tabela:

- gap_{BKS}: média do gap_{BKS} obtido pelo algoritmo, sendo este cálculo baseado na Equa-

ção (5.1), apresentada anteriormente.

- RPD: média do RPD obtido pelo algoritmo, o RPD é calculado a partir do valor obtido pelo algoritmo e do melhor valor observado dentre todos os algoritmos conforme a Equação (5.2).

$$RPD = \frac{Valor - Melhor}{Melhor} \times 100 \tag{5.2}$$

  Como é utilizado o melhor valor dentre os três algoritmos na divisão, o RPD sempre será um valor maior ou igual a 0. O RPD também foi reportado na tabela de forma complementar ao gap_{BKS} por tratar-se de um indicador frequentemente usado na literatura para comparar algoritmos para o UPMSP.

- # Melhor: o número de melhores soluções obtidas pelo algoritmo. Adicionalmente, para o algoritmo MPA FixOpt, o número de melhores soluções encontradas que superam a melhor solução previamente conhecida são apresentados entre parênteses. Caso haja novas melhores soluões esse valor é ressaltado em negrito.

Na Tabela 5.3 é possível constatar que o MPA mesmo com valores piores de gap_{BKS} e RPD, comparado com os outros algoritmo, não obteve as melhores soluções para as instâncias com 30 máquinas e 200 e 250 tarefas. Além disso, o SLS não obteve solução de melhorá para as instâncias com 200 tarefas e 10 máquinas, além das instâncias com 250 tarefas e 10 e 15 máquinas. Para todos os conjuntos com 10 máquinas, o MPA FixOpt obteve média de gap_{BKS} negativo. Além disso, das melhores soluções encontradas pelo MPA FixOpt, somente nos conjuntos com 30 máquinas e 100, 150, 200 e 250 tarefas e o conjunto com 100 tarefas e 25 máquinas, não obteve soluções melhores do que a literatura. Ademais, o MPA FixOpt foi algoritmo que possui o maior número de melhores soluções, além de obter 338 soluções melhores que literatura. Entre os algoritmos analisados, SLS foi o que obteve os melhores resultados de gap_{BKS} e RPD, em seguida vem o MPA FixOpt com diferença de $0,39\%$ no gap_{BKS} e $0,28\%$ no RPD, sendo assim valores muito próximo ao obtido pelo SLS.

Portanto, os algoritmos propostos obtiveram resultados competitivos, visto que conseguiram valores próximos a algoritmos da literatura, quando não o superam. Entre os algoritmo propostos, o algoritmo MPA FixOpt foi o que alcançou melhores resultados. Nas comparações entre este algoritmo, com a literatura, o MPA FixOpt obteve 338 soluções melhores que literatura das 1.000 instâncias de Vallada e Ruiz (2011). Além disso, o RPD obtido pelo MPA FixOpt é valor próximo do RPD obtido pelo SLS, sendo assim um algoritmo que é equiparável à algoritmos da literatura.

Tabela 5.3: Média do gap_{BKS}, RPD e quantidade de soluções ótimas para cada conjunto de 40 instâncias para os algoritmos MPA, SLS e MPA FixOpt.

Instância		MPA			SLS			MPA FixOpt		
|N|	|M|	gap_{BKS} (%)	RPD (%)	# Melhor	gap_{BKS} (%)	RPD (%)	# Melhor	gap_{BKS} (%)	RPD (%)	# Melhor
50	10	0,63	1,43	16	0,29	1,08	21	-0,78	**0,00**	40(**19**)
	15	2,71	3,12	14	0,00	**0,23**	35	0,02	0,26	35(**5**)
	20	3,24	3,85	19	0,00	**0,32**	37	0,20	0,53	33(**3**)
	25	3,01	3,34	19	0,00	**0,09**	39	0,49	0,62	35(**1**)
	30	1,81	2,09	28	0,25	**0,38**	37	0,57	0,73	35(**1**)
100	10	0,71	2,29	13	0,58	2,16	9	-1,53	**0,02**	39(**25**)
	15	5,43	7,27	6	0,04	1,12	17	-1,00	**0,08**	37(**22**)
	20	9,62	11,81	7	0,00	**0,23**	34	1,00	1,28	23(**6**)
	25	13,44	17,42	7	0,00	**0,00**	40	3,15	3,32	11(0)
	30	14,55	19,03	8	0,00	**0,00**	40	4,35	4,66	11(0)
150	10	0,01	1,93	7	1,63	3,63	2	-1,92	**0,00**	40(**33**)
	15	3,84	5,81	4	0,05	1,68	10	-1,59	**0,03**	39(**29**)
	20	9,49	11,36	7	0,00	**0,20**	33	1,07	1,32	22(**7**)
	25	16,36	21,94	4	0,00	**0,06**	39	2,60	2,80	10(**1**)
	30	21,12	30,61	6	0,00	**0,00**	40	5,85	6,32	4(0)
200	10	0,00	1,34	8	3,07	4,59	0	-1,30	**0,04**	38(**32**)
	15	2,24	5,48	1	0,29	3,27	2	-2,96	**0,00**	40(**37**)
	20	10,18	13,10	7	0,07	0,76	22	-0,45	**0,25**	35(**16**)
	25	17,89	24,26	2	0,00	**0,03**	39	3,40	3,63	6(**1**)
	30	24,63	37,23	0	0,00	**0,00**	40	6,00	6,54	5(0)
250	10	0,00	0,79	11	4,61	5,75	0	-0,76	**0,03**	36(**29**)
	15	2,76	7,06	3	0,25	4,22	0	-3,95	**0,02**	39(**37**)
	20	9,79	13,15	5	0,12	1,56	4	-1,41	**0,03**	39(**33**)
	25	22,61	42,96	3	0,00	**0,04**	39	1,95	2,08	10(**1**)
	30	30,40	60,15	0	0,00	**0,00**	40	5,29	5,74	7(0)
Média		9,06	13,95	-	0,45	**1,26**	-	0,73	1,61	-
Total		-	-	205	-	-	619	-	-	669(**338**)

Capítulo 6

Conclusões

Este trabalho abordou o Problema de Sequenciamento em Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependente da Sequência (UPMSP), que tem grande relevância prática e importância científica, dada a dificuldade em resolvê-lo. Inicialmente foram analisados os principais modelos matemáticos e algoritmos exatos da literatura para o problema. Experimentos computacionais considerando as instâncias de Vallada e Ruiz (2011) revelaram que os modelos propostos por Avalos-Rosales et al. (2015) e Fanjul-Peyro et al. (2019) tem melhor desempenho dentre todos os avaliados para resolver instâncias de pequeno porte. No entanto, para instâncias de grande porte nenhum dos modelos obteve resultados satisfatórios. Assim, este trabalho propõe heurísticas matemáticas *Fix-And-Optimize* (FixOpt) que fixam parte do problema com o objetivo de gerar subproblemas que podem ser facilmente resolvidos por métodos exatos. Foram implementados dois algoritmos: um que utiliza o modelo de Avalos-Rosales et al. (2015), com o nome "IP FixOpt", e outro que utiliza o método de Fanjul-Peyro et al. (2019), com o nome "MPA FixOpt".

Os resultados comparando os dois algoritmos propostos com o algoritmo estado-da-arte, proposto por Santos et al. (2016b), e os métodos matemáticos propostos por Avalos-Rosales et al. (2015) e Fanjul-Peyro et al. (2019), mostraram que as heurísticas matemáticas avaliadas são competitivas, sendo capazes de obter diversas soluções melhores ou iguais às melhores conhecidas na literatura. O MPA FixOpt obteve os melhores resultados dentre os dois algoritmos propostos, resultando em soluções melhores do que as da literatura para 338 das 1.000 instâncias de grande porte de Vallada e Ruiz (2011).

Para trabalhos futuros, propõe-se adaptar os algoritmos de forma a eliminar parâmetros, tornando o valor de crescimento e decrescimento do tamanho do submodelo dinâmico, por exemplo. Ademais, sugere-se combinar o SLS proposto por Santos et al. (2016b) com o MPA FixOpt, já que o SLS obteve sua convergência em menos de 10 segundos em média. Desta forma, pode-se aplicar a heurística matemática considerando uma solução inicial de melhor qualidade. Por fim, se propõe adaptar o algoritmo proposto a outras variações do problema de sequenciamento de tarefas em máquinas.

Apêndice A

Tabela

Instâncias	Novo Nome
I_50_10_S_1-124_5	I_50_10
I_50_15_S_1-99_4	I_50_15
I_50_20_S_1-49_5	I_50_20
I_50_25_S_1-9_4	I_50_25
I_50_30_S_1-9_4	I_50_30
I_100_10_S_1-124_5	I_100_10
I_100_15_S_1-49_4	I_100_15
I_100_20_S_1-124_6	I_100_20
I_100_25_S_1-124_8	I_100_25
I_100_30_S_1-9_8	I_100_30
I_150_10_S_1-99_9	I_150_10
I_150_15_S_1-49_6	I_150_15
I_150_20_S_1-9_1	I_150_20
I_150_25_S_1-124_7	I_150_25
I_150_30_S_1-9_1	I_150_30
I_200_10_S_1-49_6	I_200_10
I_200_15_S_1-99_1	I_200_15
I_200_20_S_1-124_2	I_200_20
I_200_25_S_1-49_5	I_200_25
I_200_30_S_1-124_2	I_200_30
I_250_10_S_1-124_9	I_250_10
I_250_15_S_1-9_8	I_250_15
I_250_20_S_1-49_9	I_250_20
I_250_25_S_1-124_8	I_250_25
I_250_30_S_1-49_1	I_250_30

Tabela A.1: Tabela para renomeação dos nomes das instâncias.

Referências Bibliográficas

Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17(1):177–187.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Arnaout, J.-P.; Musa, R. e Rabadi, G. (2014). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines—part ii: enhancements and experimentations. *Journal of Intelligent Manufacturing*, 25(1):43–53.

Arnaout, J.-P.; Rabadi, G. e Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21(6):693–701.

Avalos-Rosales, O.; Angel-Bello, F. e Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76(9-12):1705–1718.

Burke, E. K. e Bykov, Y. (2012). The late acceptance hill-climbing heuristic. *University of Stirling, Tech. Rep.*

Bykov, Y. e Petrovic, S. (2013). An initial study of a novel step counting hill climbing heuristic applied to timetabling problems. In *Proceedings of 6th Multidisciplinary International Scheduling Conference (MISTA 2013)*.

Chang, P.-C. e Chen, S.-H. (2011). Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft Computing*, 11(1):1263–1274.

Cota, L. P.; Guimarães, F. G.; de Oliveira, F. B. e Souza, M. J. F. (2017). An adaptive large neighborhood search with learning automata for the unrelated parallel machine scheduling problem. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pp. 185–192. IEEE.

Cota, L. P.; Haddad, M. N.; Souza, M. J. F. e Coelho, V. N. (2014). Airp: A heuristic algorithm for solving the unrelated parallel machine scheduling problem. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1855–1862. IEEE.

De Castro, L. N. e Von Zuben, F. J. (2000). The clonal selection algorithm with engineering applications. In *Proceedings of GECCO*, volume 2000, pp. 36–39.

DePuy, G. W.; Moraga, R. J. e Whitehouse, G. E. (2005). Meta-raps: a simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E: Logistics and Transportation Review*, 41(2):115–130.

Diana, R. O. M.; d. F. Filho, M. F.; d. Souza, S. R. e Silva, M. A. L. (2013). A clonal selection algorithm for makespan minimization on unrelated parallel machines with sequence dependent setup times. In *2013 Brazilian Conference on Intelligent Systems*, pp. 57–63.

Dorigo, M.; Maniezzo, V.; Colorni, A. et al. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics*, 26(1):29–41.

Fanjul-Peyro, L.; Ruiz, R. e Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research*, 101:173–182.

Fleszar, K.; Charalambous, C. e Hindi, K. S. (2012). A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 23(5):1949–1958.

Fonseca, G. H.; Santos, H. G. e Carrano, E. G. (2016). Integrating matheuristics and metaheuristics for timetabling. *Computers & Operations Research*, 74:108–117.

Glover, F. (1997). Tabu search and adaptive memory programming—advances, applications and challenges. In *Interfaces in computer science and operations research*, pp. 1–75. Springer.

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *The International Journal of Production Research*, 31(7):1579–1594.

Gurobi Optimization, L. (2018). Gurobi optimizer reference manual.

Hansen, P.; Mladenović, N. e Perez-Britos, D. (2001). Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350.

Helal, M.; Rabadi, G. e Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3(3):182–192.

Holland, J. H. (1975). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.

Holm, D. S.; Mikkelsen, R. Ø.; Sørensen, M. e Stidsen, T. R. (2019). A mip based approach for international timetabling competation 2019. *Proceedings of the International Timetabling Competition*, 2020.

ITI, I. (2019). Sistema de optimización aplicada. http://soa.iti.es/. Acessado em 04/11/2019.

Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Kurz, M. e Askin, R. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39(16):3747–3769.

Lenstra, J. K.; Kan, A. R. e Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pp. 343–362. Elsevier.

Lourenço, H. R.; Martin, O. C. e Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics*, pp. 320–353. Springer.

McConnell, S. (2004). *Code complete*. Pearson Education.

Rabadi, G.; Moraga, R. J. e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1):85–97.

Ropke, S. e Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472.

Santos, H. G.; Toffolo, T. A.; Gomes, R. A. e Ribas, S. (2016a). Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 239(1):225–251.

Santos, H. G.; Toffolo, T. A.; Silva, C. L. e Vanden Berghe, G. (2016b). Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. *International Transactions in Operational Research*.

Scheduling Research (2005). Scheduling Research. https://sites.wp.odu.edu/schedulingresearch/paper. Acessado em 12/11/2019.

Toffolo, T. A.; Santos, H. G.; Carvalho, M. A. e Soares, J. A. (2016). An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, 19(3):295–307.

Toffolo, T. A. M. e Santos, H. G. (2019). Python-MIP. http://www.python-mip.com/. Acessado em 23/09/2019.

Tran, T. T.; Araujo, A. e Beck, J. C. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):83–95.

Tran, T. T. e Beck, J. C. (2012). Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In *ECAI*, pp. 774–779.

Vallada, E. e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

Wang, L.; Wang, S. e Zheng, X. (2016). A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times. *IEEE/CAA Journal of Automatica Sinica*, 3(3):235–246.

Wauters, T.; Kinable, J.; Smet, P.; Vancroonenburg, W.; Berghe, G. V. e Verstichel, J. (2016). The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 19(3):271–283.

Ying, K.-C.; Lee, Z.-J. e Lin, S.-W. (2012). Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 23(5):1795–1803.