FELIPPE MOREIRA FAÊDA

MÉTODOS DE RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM RESTRIÇÕES DE PRECEDÊNCIA E TEMPOS DE PREPARAÇÃO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2015

Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa

T
F147m
2015

Faêda, Felippe Moreira, 1990-
Métodos de resolução do problema de sequenciamento em máquinas paralelas não-relacionadas com restrições de precedência e tempos de preparação / Felippe Moreira Faêda. – Viçosa, MG, 2015.
xiii, 98f : il. (algumas color.) ; 29 cm.

Inclui apêndices.
Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.88-91.

1. Otimização combinatória. 2. Algoritmos. 3. Heurística. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22. ed. 519.64

FELIPPE MOREIRA FAÊDA

MÉTODOS DE RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM RESTRIÇÕES DE PRECEDÊNCIA E TEMPOS DE PREPARAÇÃO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 10 de dezembro de 2015.

André Gustavo dos Santos

Martin Gómez Ravetti

José Elias Claudio Arroyo
Orientador

Dedico este trabalho aos meus pais Rafael e Leila, aos meus irmãos Leonardo e Ana Clara, e a minha noiva Josiane.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela minha vida, pela minha saúde e pela minha família. Agradeço por estar sempre ao meu lado dando forças e coragem para superar os desafios.

Agradeço aos meus pais, Rafael e Leia, pelo apoio e educação que me deram. Agradeço imensamente a eles por sempre estarem ao meu lado, buscando meu bem estar, saúde e educação. O apoio e incentivo deles foram fundamentais para essa conquista.

Agradeço aos meus irmãos, Leonardo e Ana Clara, pelo carinho e amizade. Agradeço a minha vó Aparecida pelas orações e cuidado que tem comigo. Também agradeço a minha vó Maria pelo amor e carinho que me deu em vida e sei que onde estiver estará feliz por mim. E a todos os familiares pelas palavras de incentivo.

Agradeço também a minha noiva Josiane pelo carinho, companheirismo e paciência durante o período do mestrado.

Agradeço todos os amigos do Departamento de Informática, especialmente ao Ítalo e Renan pelo companheirismo e amizade.

Agradeço a todos os professores do Departamento de Informática da UFV que contribuíram em minha formação. Especialmente, ao meu orientador José Elias Arroyo e ao professor André Gustavo pela atenção e pelos ensinamentos. Obrigado pela oportunidade de viajar com vocês para a Itália, onde conhecemos a universidade de Bolonha, foi uma experiência que vou levar para toda vida. Agradeço ao professor Michele Monaci pela receptividade em seu país e pela contribuição na pesquisa.

Obrigado a todos vocês.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

FAÊDA, Felippe Moreira, M.Sc., Universidade Federal de Viçosa, Dezembro de 2015. **Métodos de Resolução do Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Restrições de Precedência e Tempos de Preparação.** Orientador: José Elias Claudio Arroyo.

Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas considerando restrições de precedência entre as tarefas e tempos de preparação dependentes da sequência e da máquina. Este problema tem como objetivo minimizar o tempo máximo de conclusão do sequenciamento, conhecido como *makespan*. Em problemas que consideram restrições de precedência, nenhuma tarefa pode iniciar seu processamento sem que todas as suas tarefas predecessoras tenham sido concluídas. Para resolver este problema foram desenvolvidos três modelos de programação linear inteira mista (PLIM), denotados por Modelo 1, Modelo 2 e Modelo 3. Em seguida, sete heurísticas construtivas foram desenvolvidas, denotadas por $HC1$ a $HC7$, as quais se diferenciam pelas regras de prioridade utilizadas. Neste trabalho também é implementado o método chamado Proximity Search (PS), que tenta determinar soluções ótimas para o problema. O método PS precisa de uma solução inicial e de um modelo base de PLIM. Neste método a função objetivo do modelo é substituída por uma função de proximidade e o conjunto de soluções viáveis é reduzido através da adição de cortes. A ideia é, iterativamente, resolver o modelo com a tentativa de melhorar a solução corrente. Foram desenvolvidas três versões do PS denotadas por $PS1$, $PS2$ e $PS2_{RINS}$. Neste trabalho também foram desenvolvidos algoritmos baseados em meta-heurísticas a fim de resolver o problema de forma aproximada. Primeiramente, foram desenvolvidas duas buscas locais denotadas por BL1 e BL2 baseadas na estratégia de inserção por vizinhança. Em seguida, foram implementadas duas meta-heurísticas: GRASP (*Greedy Randomized Adaptive Search*) e IG (*Iterated Greedy*). Experimentos computacionais e

análises estatísticas foram realizados a fim de comparar o desempenho dos modelos, das versões do PS e das heurísticas propostas. De acordo com os experimentos, o Modelo 1 apresentou-se mais eficiente na qualidade das soluções obtidas e a heurística $HC7$ mostrou-se mais eficiente na geração de uma solução razoavelmente boa. Além disso, as versões do PS obtiveram melhorias na qualidade da solução obtida e redução no tempo computacional gasto se comparado ao Modelo 1. Em seguida, o IG obteve desempenho significativamente melhor que o GRASP e o PS em relação à qualidade da solução final e a velocidade com que a solução corrente é melhorada.

Abstract

FAÊDA, Felippe Moreira, M.Sc., Universidade Federal de Viçosa. December, 2015. **Resolution methods for the unrelated parallel machine sequeduling problem with precedence constraints and setup times.** Adviser: José Elias Claudio Arroyo.

In this work we address the scheduling problem in unrelated parallel machine with precedence constraints between the jobs and sequence-dependent and machine-dependent setup times. The objective of this problem is to minimize the maximum completion time of sequence, called makespan. The precedence constraints force a job not to be started before all its predecessors are finished. To solve this problem, we developed three models of mixed integer programming (MIP), denoted by Model 1, Model 2 and Model 3. Next, seven constructive heuristics were developed, denoted by $HC1$ to $HC7$, which differ in the priority rules. Also in this work, a method called Proximity Search (PS) is implemented, which tries to find optimal solutions to the problem. The method requires an initial solution and a MILP-based model. In this method, the objective function of the model is replaced by a proximity function and the set of feasible solutions is reduced by the addition of cuts. The idea is to iteratively solve the model trying to improve the current solution. We developed three versions of the PS denoted by $PS1$, $PS2$ and $PS2_{RINS}$. In addition, we developed algorithms based on metaheuristics to solve the problem approximately. First, were developed two local searches denoted by $BL1$ and $BL2$ based on the insertion neighborhood. Next, were implemented two metaheuristics: GRASP (*Greedy Randomized Adaptive Search*) and IG (*Iterated Greedy*). Computational experiments and statistical analyzes were performed in order to compare the performance of models, PS versions and heuristics. According to the experiments, the Model 1 is more efficient in the quality of solutions and the HC7 heuristic is more efficient in generating a reasonably good solution. In addition, the versions of the

PS obtained improvements in the quality of the obtained solution and reduction in computational time spent compared to Model 1. Then, the IG obtained significantly better performance than the GRASP and PS in relation to the quality of the final solution and the speed with which the current solution is improved.

Capítulo 1

Introdução

Com o crescimento econômico mundial e com o aumento da globalização nas últimas décadas, criou-se de um cenário competitivo entre as indústrias de manufatura e de serviços. Além disso, a exigência por uma maior eficiência produtiva utilizando de forma adequada os recursos disponíveis, fez com que as empresas realizassem altos investimentos e aperfeiçoamentos nos processos de produção. Neste contexto, a programação da produção torna-se essencial para a sobrevivência de muitas empresas. Segundo Pinedo (2012), a programação da produção é um processo de tomada de decisões muito utilizado por indústrias de manufaturas e serviços que desejam alocar as tarefas aos recursos disponíveis a fim de cumprir determinados critérios de produção.

Uma das principais atividades da programação da produção é o sequenciamento de tarefas, também conhecido como *scheduling*. O sequenciamento de tarefas tem como objetivo alocar um conjunto de n tarefas em um conjunto de m máquinas disponíveis determinando a ordem de processamento com que as tarefas são executadas por cada máquina, visando alcançar um ou mais critérios de otimização. Estes critérios estão relacionados principalmente à utilização eficiente de recursos, redução do tempo total de produção, cumprimento dos prazos de entrega combinados com os clientes, diminuição dos custos com a produção, estocagem e entregas dos produtos, e entre outros critérios.

Neste trabalho é abordado o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas considerando restrições de precedência entre as tarefas e tempos de preparação dependentes da sequência e da máquina. O problema consiste em sequenciar um conjunto de n tarefas, $J = \{1, ..., n\}$, em um conjunto de m máquinas, $M = \{1, ..., m\}$. Estas máquinas são consideradas não-relacionadas por possuírem diferentes velocidades de processamento, portanto, cada

tarefa $j \in J$ tem um tempo de processamento distinto para cada máquina $i \in M$. Além disso, os tempos de preparação ou *setup times* são considerados dependentes da sequência e dependentes das máquinas, ou seja, o tempo de preparação será o tempo gasto no preparo de uma máquina i a fim de executar uma tarefa k alocada imediatamente após a uma tarefa j. Considerando a restrição de precedência entre as tarefas, uma tarefa $j \in J$ somente pode começar seu processamento se todas as suas tarefas predecessoras tiverem sido finalizadas. O objetivo é sequenciar as n tarefas nas m máquinas a fim de minimizar o tempo máximo de conclusão das tarefas, conhecido como *makespan*. De acordo com a classificação de Graham et al. (1979), o problema pode ser denotado como $R_m \mid prec, S_{ijk} \mid C_{max}$, onde R_m representa as máquinas paralelas não-relacionadas, *prec* a restrição de precedência, S_{ijk} o tempo de preparação dependente da máquina e da sequência, e o C_{max} que representa o *makespan*.

Gary & Johnson (1979) provam que o problema básico de sequenciamento em máquinas paralelas idênticas ($P_m \mid\mid C_{max}$) é um problema NP-difícil a partir de $m = 2$ máquinas. Como o problema $R_m \mid prec, S_{ijk} \mid C_{max}$ é uma generalização do problema $P_m \mid\mid C_{max}$, então também é um problema NP-difícil. Portanto, não se conhece algoritmos eficientes que resolva o problema de forma exata em um tempo computacional aceitável. Com isso, é viável aplicar métodos heurísticos para resolvê-los de forma aproximada e em tempo computacional aceitável. Por essa característica, o problema possui grande importância teórica devido a possibilidade de desenvolver diferentes métodos heurísticos para resolução do problema.

O problema também possui uma importância prática por estar presente no cenário de muitas indústrias de manufatura e serviço. Segundo Rabadi et al. (2006), o problema de sequenciamento em máquinas paralelas não-relacionadas considerando tempos de preparação é muito comum em indústrias de tecido, vidro, semicondutores, produtos químicos e algumas indústrias de serviço. Liu (2013) também cita que o problema considerando restrições de precedência ocorre tipicamente em escritórios ou ambientes de gerenciamento de projetos. Hassan Abdel-Jabbar et al. (2014) também trabalham com restrições de precedência em máquinas paralelas, porém, aplicadas ao contexto de computação em nuvem.

Na literatura não foram encontrados trabalhos que consideram as mesmas características abordadas neste trabalho. Portanto, a ausência de trabalhos na literatura, a complexidade computacional e a importância prática do problema, tornam-se as principais motivações para este trabalho.

Primeiramente, neste trabalho aborda-se o desenvolvimento de modelos de programação matemática específicos para o problema. Em seguida, visando melho-

rar o desempenho da resolução dos modelos matemáticos utilizando métodos exatos, também é aplicado um recente método de resolução chamado de *Proximity Search*, proposto por Fischetti & Monaci (2014). Por se tratar de um problema de otimização combinatória considerado difícil de resolver, a utilização de métodos heurísticos é, geralmente, uma boa opção para encontrar soluções de boa qualidade em tempo aceitável. Portanto, neste trabalho são desenvolvidas diferentes heurísticas construtivas baseadas em regras de prioridade. Em seguida, são desenvolvidas duas heurísticas de melhoramento: uma baseada na meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), inicialmente proposta por Feo & Resende (1995), e a outra baseada na meta-heurística *Iterated Greedy* (IG) proposta por Ruiz & Stützle (2007). As heurísticas propostas são analisadas e comparadas estatisticamente sobre um conjunto de instâncias geradas para o problema a fim de provar seu bom desempenho.

1.1 Objetivos

O objetivo deste trabalho é desenvolver algoritmos baseados em técnicas de resolução exatas e heurísticas que sejam eficientes na produção de soluções de alta qualidade em tempo computacional aceitável, a fim de resolver o problema de sequenciamento em máquinas paralelas não-relacionadas com restrição de precedência entre as tarefas e tempos de preparação dependentes da sequência e da máquina.

1.1.1 Objetivos Específicos

Para alcançar o objetivo geral é necessário que sejam atingidos os objetivos específicos a seguir:

- Analisar e estudar os trabalhos da literatura que abordam problemas relacionados, obtendo assim os conceitos das principais estratégias de resolução aplicadas a problemas semelhantes;
- Implementar diferentes modelos de programação matemática para o problema proposto;
- Desenvolver um algoritmo que utilize uma estratégia eficiente de busca aproximada aplicando técnicas de resolução exatas;
- Desenvolver algoritmos que implementem diferentes heurísticas construtivas e eficientes meta-heurísticas.

- Realizar experimentos computacionais utilizando instâncias geradas para o problema a fim de comprovar e comparar a eficiência dos algoritmos desenvolvidos;
- Analisar os resultados obtidos e validar o desempenho dos métodos através de testes estatísticos.

1.2 Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte maneira: no capítulo 2 é apresentada a caracterização do problema e uma revisão bibliográfica dos trabalhos que tratam de problemas em máquinas paralelas com restrição de precedência. O capítulo 3 aborda as principais características dos métodos de resolução de problemas de otimização aplicados neste trabalho. O capítulo 4 apresenta a formulação de três modelos matemáticos propostos para o problema e seus respectivos experimentos computacionais. Nos capítulos 5, 6 e 7 são apresentadas, respectivamente, as heurísticas construtivas, o *Proximity Search* e as meta-heurísticas aplicadas. Além disso, são discutidos nestes capítulos os experimentos computacionais realizados para cada uma das heurísticas propostas. Finalmente, no capítulo 8 são apresentadas as considerações finais e as propostas de trabalhos futuros.

Capítulo 2

Sequenciamento de Tarefas em Máquinas Paralelas com Restrição de Precedência

2.1 Máquinas Paralelas com Restrição de Precedência

Na literatura são encontrados trabalhos que consideram três diferentes tipos de máquinas paralelas, que são identificadas como: idênticas, uniformes e não-relacionadas.

Em máquinas paralelas idênticas, as máquinas possuem as mesmas características em termos de velocidade de processamento, ou seja, as tarefas são executadas com mesmo tempo independente da máquina. Já em máquinas paralelas uniformes ou também denominadas de máquinas paralelas relacionadas, cada máquina i possui uma velocidade específica q_i e cada tarefa j possui um tempo de processamento p_j. A execução de uma tarefa j em uma máquina uniforme i requer $\frac{p_j}{q_i}$ unidades de tempo. Em máquinas paralelas não-relacionadas, as máquinas também possuem diferentes velocidades de processamento, por isso, cada tarefa j possui um tempo de processamento p_{ij} diferente para cada máquina i. No problema abordado neste trabalho será considerada máquinas paralelas não-relacionadas.

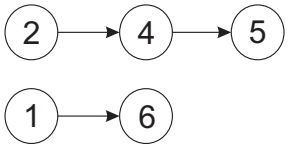
Na literatura também podem ser encontradas diferentes variações nas restrições de precedência. As principais variações encontradas são conhecidas como: precedência arbitrária, *s-precedence*, em cadeia, floresta, *out-tree* e *in-tree*.

Na restrição de precedência arbitrária, a relação de precedência entre cada

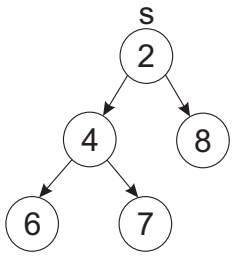
dois pares de tarefas j e k é determinada por uma função de probabilidade. Na restrição de precedência denominada *s-precedence*, a existência de precedência entre duas tarefas j e k obriga que a tarefa k somente inicie seu processamento quando a tarefa j tiver sido inicializada, que é diferente da restrição de precedência padrão, onde a tarefa k não pode iniciar até que a tarefa j tenha sido concluída (Kim et al., 2009).

As restrições de precedência do tipo cadeia, floresta, *out-tree* e *in-tree* se diferenciam na forma em que o grafo de precedência é criado. O grafo de precedência é denotado por D = (V, A), onde V é formado pelos vértices que representam as tarefas e A é composto pelas arestas que representam cada relação de precedência. Por exemplo, na Figura 2.1(a) a tarefa 2 precede a tarefa 4, e a tarefa 4 precede a tarefa 5.

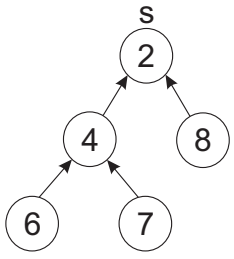
(a) Cadeia.



(b) Floresta.



(c) Out-Tree.



(d) In-Tree.

Figura 2.1. Tipos de Restrições de Precedência

Em problemas com restrição de precedência em cadeia, cada tarefa das pre-

cedências existentes podem ter no máximo uma tarefa predecessora e no máximo uma tarefa sucessora. Portanto, as precedências formam um conjunto de tarefas em cadeia, conforme pode ser observado na Figura 2.1(a). Por exemplo, se a tarefa 2 apresentada na figura tiver sido concluída, a próxima tarefa da cadeia, que é a tarefa 4, está liberada para ser processada (HERRMANN ET AL., 1997).

A restrição de precedência em floresta possui esse nome devido ao grafo de precedência ser do tipo floresta. Uma floresta é um grafo acíclico não conexo. Nesta restrição, a floresta pode ser considerada um conjunto de grafos em cadeia, de acordo com a Figura 2.1(b). Já as restrições de precedência *out-tree* são formadas por uma árvore T do tipo *out-tree*. Como pode ser visto na Figura 2.1(c), esta árvore possui um nó raiz s e o único caminho entre s e cada vértice de T é um caminho direcionado. Já as restrições de precedência *in-tree* são formadas por uma árvore T do tipo *in-tree*. Conforme Figura 2.1(d), esta árvore possui um nó raiz s e o único caminho entre cada vértice de T e s é um caminho direcionado.

No problema abordado neste trabalho será considerado restrição de precedência arbitrária. Em cada uma das próximas seções são apresentados trabalhos da literatura para cada tipo de máquina. Além disso, será apresentado o tipo de restrição de precedência considerado em cada trabalho.

2.2 Caracterização do Problema Abordado

O problema $R_m \mid prec, S_{ijk} \mid C_{max}$ é caracterizado por possuir um conjunto de tarefas $J = \{1, ..., n\}$ com um total de n tarefas e um conjunto de máquinas $M = \{1, ..., m\}$ com um total de m máquinas, cujo objetivo é alocar as n tarefas nas m máquinas de forma a minimizar o tempo máximo de conclusão das tarefas, também conhecido como *makespan* (C_{max}).

O problema precisa respeitar as seguintes características:

- Cada tarefa $j \in J$ deve ser processada uma única vez, sem interrupções e por apenas uma única máquina $i \in M$;
- As máquinas são classificadas como máquinas paralelas não-relacionadas, ou seja, cada máquina possui diferentes velocidades de processamento. Portanto, cada tarefa j possui um tempo de processamento específico p_{ij} para cada máquina i.
- Também são considerados os tempos de preparação, denotado por S_{ijk}. O tempo de preparação é o tempo gasto para preparar uma máquina i para pro-

cessar uma tarefa k alocada imediatamente após a tarefa j. Os tempos de preparação são dependentes da sequência, ou seja, uma tarefa k pode possuir diferentes tempos de preparação dependendo da ordem com que está alocada ($S_{ijk} \neq S_{ikj}$). Além disso, os tempos de preparação são dependentes da máquina, ou seja, uma tarefa k pode possuir diferentes tempos de preparação dependendo da máquina onde está alocada ($S_{ijk} \neq S_{hjk}$).

- O problema abordado também considera a restrição de precedência entre as tarefas. Nesta restrição, é dado um conjunto de precedências $A = \{a_1, a_2, ..., a_i, ..., a_p\}$ com p relações de precedência, onde a i-ésima restrição $a_i = (j, k)$ representa que a tarefa j deve preceder a tarefa k. Portanto, o tempo de início do processamento da tarefa k não deve ser menor que o tempo de conclusão da tarefa j. Essa restrição garante que uma tarefa não inicie seu processamento até que todas as suas tarefas predecessoras tenham sido finalizadas.

Para mostrar as características do problema em estudo, será apresentado a seguir um exemplo de sequenciamento para uma instância com $n = 8$ tarefas e $m = 2$ máquinas. Os tempos de processamento das tarefas em cada máquina são descritos na Tabela 2.1. Nas Tabelas 2.2 e 2.3 são descritos os tempos de preparação das tarefas para as máquinas M_1 e M_2, respectivamente.

Na Tabela 2.4 estão todas relações de precedência entre as tarefas. Esta Tabela possui todo o conteúdo do conjunto de precedências A, onde j representa a tarefa predecessora e k a tarefa sucessora. De acordo com a restrição de precedência, a tarefa j precede obrigatoriamente a tarefa k, ou seja, a tarefa k somente pode iniciar seu processamento quando a tarefa j estiver concluída. Considerando as tarefas contidas no conjunto de precedências A como sendo vértices de um grafo, e as relações de precedência como sendo as arestas, o grafo formado pelo conjunto A, chamado de grafo de precedência, pode ser observado pela Figura 2.2. Como exemplo, pode ser observado no grafo de precedência que a tarefa 2 precede a tarefa 3, portanto, a tarefa 3 só pode ser processada quando a tarefa 2 for finalizada. É importante notar que para uma instância do problema ser válida, o grafo de precedência não pode conter ciclos.

Utilizando as informações contidas nas tabelas 2.1, 2.2, 2.3 e 2.4, uma possível solução para o exemplo é apresentada na Figura 2.3, onde o sequenciamento é ilustrado através de um gráfico de Gantt. Como pode ser observado, na máquina M_1 estão sequenciadas as tarefas 4, 2, 8 e 7, nesta ordem. Na máquina M_2 estão sequenciadas as tarefas 1, 6, 5 e 3, também nesta ordem. No gráfico, os retângulos na cor

Tabela 2.1. Tempos de Processamento das Tarefas nas Máquinas M_1 e M_2

	M_1	M_2
1	35	24
2	45	51
3	63	14
4	39	72
5	17	21
6	27	12
7	14	78
8	7	90

Tabela 2.2. Tempos de Preparação na Máquina M_1

S_{1jk}	1	2	3	4	5	6	7	8
0	3	4	0	3	5	5	3	2
1	6	1	6	3	6	3	5	1
2	4	0	6	2	7	4	5	2
3	3	3	6	3	8	9	6	6
4	2	5	7	6	8	2	9	0
5	0	9	2	6	8	4	9	6
6	9	5	1	2	0	0	5	2
7	5	9	3	9	7	0	2	9
8	5	6	2	5	4	7	2	1

Tabela 2.3. Tempos de Preparação na Máquina M_2

S_{2jk}	1	2	3	4	5	6	7	8
0	6	3	3	0	6	5	5	2
1	1	2	0	8	1	4	9	2
2	2	8	1	5	0	5	2	8
3	4	3	7	4	4	7	5	5
4	4	8	0	5	0	3	1	8
5	8	1	2	3	4	0	9	4
6	9	6	3	9	3	3	2	6
7	0	5	7	5	6	2	0	2
8	2	8	9	5	5	6	4	8

Tabela 2.4. Relação de precedência entre as tarefas

A	j	k
a_1	2	3
a_2	2	8
a_3	3	7
a_4	4	6
a_5	5	8

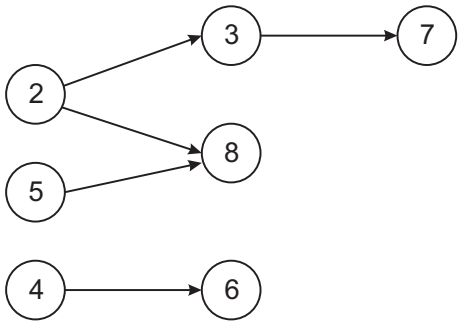


Figura 2.2. Grafo de Precedência

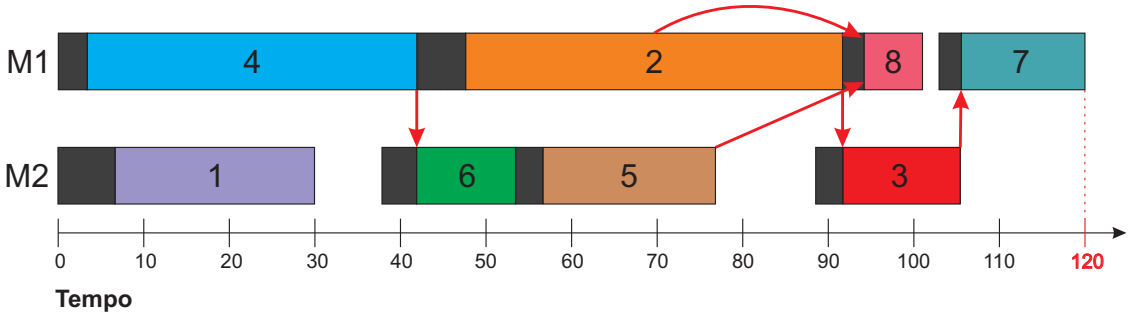


Figura 2.3. Exemplo de um possível sequenciamento para a instância.

preta representam os tempos de preparação e os outros retângulos representam as tarefas, sendo que os tamanhos das bases representam os tempos de processamento.

Como pode ser observado no gráfico da Figura 2.3, as setas indicam as precedências entre as tarefas. Além disso, existem intervalos ociosos de tempo entre as tarefas 1 e 6, 5 e 3, e, 8 e 7. Esses intervalos são provocados pelas relações de precedência. Como por exemplo, o tempo de início da tarefa 6 poderia ser logo após o tempo de conclusão da tarefa 1 na máquina M_2, no entanto, é necessário atrasar o início do processamento da tarefa 6, pois ela depende da tarefa 4 ser concluída. Caso ocorra este intervalo ocioso, a preparação da máquina pode ser realizada dentro deste intervalo, como acontece com a tarefa 6 na máquina M_2.

Sendo E o maior tempo de conclusão das tarefas predecessoras de uma tarefa k, e j a tarefa antecessora imediata à tarefa k na máquina i, então o tempo de conclusão da tarefa k (C_k) pode ser obtido a partir da seguinte equação:

$$C_k = max\{E, (C_j + S_{ijk})\} + p_{ik} \tag{2.1}$$

Como exemplo, pode-se calcular o tempo de conclusão da tarefa 6 na máquina M_2 da seguinte forma:

$$C_6 = max\{C_4, (C_1 + S_{216})\} + p_{26} \tag{2.2}$$

$$C_6 = max\{42, (30 + 4)\} + 12 = 54 \tag{2.3}$$

É importante notar que para calcular o tempo de conclusão de uma tarefa, é necessário conhecer previamente o tempo de conclusão de todas as suas tarefas predecessoras. Conhecendo-se o tempo de conclusão de todas as tarefas sequenciadas, o *makespan* será o tempo de conclusão da última tarefa a concluir seu processamento, no caso deste exemplo da Figura 2.3, a tarefa 7 alocada na máquina M_1 é a última tarefa a concluir seu processamento, com tempo de conclusão igual a 120, sendo esse o valor do *makespan*.

2.3 Revisão Bibliográfica

2.3.1 Máquinas Paralelas Idênticas

Em Ramachandra & Elmaghraby (2006) é abordado o problema de sequenciamento de tarefas considerando apenas duas máquinas paralelas idênticas e restrição de precedência arbitrária. O objetivo é minimizar o tempo de conclusão total ponderado. O problema pode ser definido pela seguinte notação: $P_2 \mid Prec \mid \sum w_j C_j$. Os autores inicialmente propõem um modelo de programação inteira binária (BIP) e um algoritmo de programação dinâmica (DP). O algoritmo DP encontrou quase o dobro de soluções ótimas para as instâncias do problema em relação ao BIP, no entanto, o algoritmo atinge o tempo computacional limite para instâncias com até 25 tarefas. Os autores também propõem um algoritmo genético (GA) que demonstrou ser capaz de resolver problemas de qualquer tamanho e com mais de duas máquinas. Os autores conseguiram gerar uma boa população inicial para o GA, o que fez com que o algoritmo encontrasse soluções ótimas do problema com poucas iterações.

O trabalho de Queyranne & Schulz (2006) trata o problema de sequenciamento em máquinas idênticas onde é considerado um tempo de atraso entre as precedências (*precedence delays*) e *release time* das tarefas. Este problema é denotado por $P \mid r_j, Prec \mid \sum w_j C_j$. A cada restrição de precedência está associada uma quantidade de tempo, que é o tempo em que uma tarefa k deve aguardar para iniciar seu processamento após a conclusão da sua tarefa predecessora j. O objetivo é minimizar o tempo de conclusão total ponderado. Para resolver o problema, os autores desenvolveram alguns algoritmos de aproximação que se mostram eficazes

e melhores que outros algoritmos de aproximação da literatura. Estes algoritmos foram aplicados à diferentes variações deste problema, onde também se mostraram eficientes em sua resolução.

Em Kim et al. (2009) é abordado o problema de sequenciamento em máquinas paralelas idênticas com o objetivo de minimizar o tempo total de conclusão das tarefas. Este trabalho considera a restrição de precedência denominada *s-precedence*. O problema é denotado por $P \mid s\text{-}prec \mid C_j$. Os autores propõem uma formulação de programação linear (LP) e uma heurística construtiva baseada na resolução do problema de LP proposto.

No trabalho de Gacias et al. (2010), o problema de sequenciamento considera restrição de precedência arbitrária, tempos de preparação dependentes somente da sequência e *release time*. Este trabalho aborda dois objetivos: um é minimizar o tempo de conclusão total, representado pela notação $P_m \mid prec, S_{ij}, r_i \mid \sum C_i$, e o outro é minimizar o tempo de atraso máximo, representado pela notação $P_m \mid prec, S_{ij}, r_i \mid L_{max}$. Os autores propõem um algoritmo exato *Branch-and-Bound*, apresenta algumas regras de dominância e desenvolve um método de busca em árvore conhecido como *Climbing Discrepancy Search* (CDS). Os algoritmos foram comparados com outras propostas similares da literatura, onde se demonstrou mais eficiente para maior parte das instancias geradas.

Em Driessel & Mönch (2011) é tratado o problema de sequenciamento em máquinas idênticas, considerando restrição de precedência arbitrária, tempo de preparação dependente da sequência e *ready times* das tarefas. O objetivo é minimizar o tempo total de atraso ponderado. O problema possui a seguinte notação: $P_m \mid r_j, S_{ij}, prec \mid \sum w_j T_j$. Os autores desenvolveram algumas variações da metaheurística chamada *variable neighborhood search* (VNS) e comparou seu desempenho com uma heurística construtiva usando a regra de despacho *Apparent Tardiness Cost with Setups and Ready Times* (ATCSR). Os resultados mostraram um bom desempenho do VNS em relação à heurística ATCSR e obtiveram resultados melhores com a heurística ATCSR como solução inicial para o VNS.

No trabalho de Yuan et al. (2012) é considerado o problema de sequenciamento de tarefas em duas máquinas paralelas idênticas com restrição de precedência em cadeia, cujo objetivo é minimizar o *makespan*. No começo do processamento, nem todas as tarefas existem, por isso, elas vão surgindo ao decorrer do tempo. Essa característica é conhecida como *online scheduling*. Além disso, as tarefas possuem os mesmos tempos de processamento. O problema pode ser denotado por $P_2 \mid online, chains, p_j = p \mid C_{max}$. Os autores desenvolveram um algoritmo, chamado de *DelayTwice*, que resolve o problema proposto considerando a característica

online scheduling. Como provado por Huo & Leung (2005), é impossível obter um algoritmo com taxa de competitividade $p = 1$ para o problema. No entanto, o algoritmo proposto obteve a melhor taxa de competitividade para o problema, que é igual a $\frac{\sqrt{13}-1}{2}$.

2.3.2 Máquinas Paralelas Uniformes

Em Brucker et al. (1999) é tratado o problema de sequenciamento de tarefas em duas máquinas paralelas uniformes com restrição de precedência em cadeia cujo objetivo é minimizar o *makespan*. As tarefas são consideradas idênticas, pois possuem o mesmo tempo de processamento $p_j = 1$. Para resolver o problema denotado por $Q_2 \mid chains, p_j = 1 \mid C_{max}$, os autores desenvolveram um algoritmo que resolve o problema em tempo linear. Não foi possível resolver em tempo linear problemas com mais de duas máquinas e com restrição de precedência baseada em árvores.

Em Woeginger (2000) é tratado o problema de sequenciamento de tarefas em máquinas paralelas uniformes com restrição de precedência em cadeia com o objetivo de minimizar o *makespan*. O problema pode ser denotado por $Q \mid chains \mid C_{max}$. O autor propõe um simples algoritmo de aproximação resolvido em tempo polinomial. Em um algoritmo de aproximação é encontrada uma solução s próxima da solução ótima s_{opt}. Se o custo da solução s é quase sempre um fator p em relação ao custo de s_{opt}, então o algoritmo é chamado de *p-approximation*. No caso do algoritmo proposto, o algoritmo possui $p = 2$, portanto, ele é denominado *2-approximation algorithm*, que é um bom resultado para o problema.

No trabalho de Kim (2011) é considerado o problema de sequenciamento de tarefas em máquinas paralelas uniformes com restrição de precedência *s-precedence*. O objetivo é minimizar o tempo de conclusão total ponderado. O problema é denotado por $Q \mid s\text{-}prec \mid \sum w_j C_j$. Uma heurística construtiva baseada na resolução de um problema de programação linear (PL) é desenvolvida para o problema. A heurística utiliza como parâmetro a solução do PL formulada para o problema abordado, no entanto, permitindo preempção. A técnica de plano de corte é adaptada para resolver o PL eficientemente. Os resultados mostram que a heurística encontra boas soluções dentre de um curto espaço de tempo.

2.3.3 Máquinas Paralelas Não-Relacionadas

Em Herrmann et al. (1997) é abordado o problema de sequenciamento em máquinas paralelas não-relacionadas considerando restrição de precedência em cadeia e mini-

mização do *makespan*. O problema pode ser denotado por $R_m \mid chain \mid C_{max}$. Neste trabalho são propostas diferentes heurísticas construtivas para o problema a fim de encontrar boas soluções. Os resultados mostraram que as heurísticas são capazes de encontrar boas soluções aproximadas.

Em Kumar et al. (2009) é apresentado um algoritmo de aproximação considerando restrição de precedência em floresta e aplicando o algoritmo para dois objetivos: minimização do *makespan* e a minimização do tempo de conclusão total ponderado, representados pela notação $R_m \mid forest \mid C_{max}$ e $R_m \mid forest \mid \sum w_j C_j$, respectivamente.

Em Liu & Yang (2011) é abordado o problema de sequenciamento em máquinas paralelas não-relacionadas com restrição de precedência arbitrária cujo objetivo é minimizar o *makespan*, denotado por $R_m \mid prec \mid C_{max}$. Neste trabalho é proposta uma heurística construtiva baseada em regras de prioridade, denotada por SS (*Heuristic Serial Schedule Algorithm*). O algoritmo SS possui tempo polinomial $O(n^2)$. A cada iteração do algoritmo é aplicada uma regra de prioridade que utiliza a média aritmética e o desvio dos tempos de processamento para determinar uma tarefa j a ser processada por uma máquina i. O algoritmo SS se demonstrou eficaz na busca por soluções de boa qualidade utilizando pouco tempo computacional.

No trabalho de Nouri & Ghodsi (2012) é estudado o problema de sequenciamento de n tarefas em um conjunto de w trabalhadores (máquinas) não-relacionados. Cada tarefa pode ser atribuída a mais de um trabalhador. O problema considera restrição de precedência arbitrária e o objetivo é minimizar o tempo total de espera. O tempo de processamento de uma tarefa executada por um trabalhador não é fixo, cada trabalhador pode executar uma tarefa dentro de um prazo determinado por uma distribuição exponencial. Os autores desenvolveram um algoritmo que se baseia na redução de um problema de programação linear inteira 0-1 também proposto por eles. O algoritmo encontra soluções ótimas para o problema em tempo polinomial $O(mdn^d log(nd))$, sendo m o número de trabalhadores, n o número de tarefas e d um valor constante que representa a largura do grafo de precedência.

Em Liu (2013) trata-se do problema de sequenciamento em máquinas paralelas não-relacionadas com restrição de precedência arbitrária com o objetivo de minimizar o atraso total. O problema possui a seguinte notação: $R_m \mid prec \mid T$. Para resolver o problema é proposta uma heurística construtiva baseada em regras de prioridade denotada por PRHA. Além disso, é proposto um algoritmo genético híbrido (HGA) onde metade da população inicial é formada por um conjunto de soluções geradas de forma aleatória e, a outra metade, pela heurística construtiva PRHA. O algoritmo HGA apresentou-se mais eficiente para resolver instâncias pequenas do

que o modelo matemático resolvido pelo CPLEX, e, para instâncias maiores, o HGA demonstrou melhor desempenho que o algoritmo genético tradicional.

Em Hassan Abdel-Jabbar et al. (2014) é tratado o problema de sequenciamento de tarefas aplicado ao cenário de computação em nuvem. O problema compara as máquinas virtuais (VM) responsáveis por manter o serviço de computação em nuvem como máquinas paralelas não-relacionadas, considera restrição de precedência arbitrária entre as tarefas e tem como objetivo minimizar o *makespan*. O problema pode ser denotado por $R_m \mid prec \mid C_{max}$. Neste trabalho é proposto um simples algoritmo genético (GA). Nos experimentos realizados, o GA foi comparado com uma heurística construtiva da literatura e com um modelo matemático de programação linear inteira. Os resultados mostram que o GA proposto é um algoritmo eficiente, capaz de melhorar as soluções iniciais geradas e possui melhores resultados que a heurística construtiva e o modelo.

Os trabalhos apresentados nesta seção abordam problemas com diferentes características, se diferenciando principalmente no tipo de restrição de precedência, nos tipos de máquinas paralelas e nos objetivos a serem alcançados. Não foram encontrados trabalhos na literatura que utilizem as mesmas características consideradas para o problema abordado nesta dissertação.

Capítulo 3

Métodos para Resolução de Problemas de Otimização Combinatória

Os métodos utilizados para resolver problemas de otimização combinatória são divididos em dois tipos: os métodos exatos e os métodos heurísticos.

Os métodos exatos tem como objetivo encontrar a solução com o melhor valor de função objetivo de todas as soluções do espaço de busca e garantir sua otimalidade. Esta solução é definida como um ótimo global. Dentre os métodos exatos, podem-se destacar os métodos: *branch-and-bound*, *branch-and-cut*, *branch-and-price*, *cutting plane*, geração de colunas, programação dinâmica, entre outros.

Ao contrário dos métodos exatos, nos métodos heurísticos não é possível garantir a otimalidade de uma solução e nem definir o quanto uma solução está próxima da solução ótima. No entanto, os métodos heurísticos produzem soluções de boa qualidade consumindo um tempo computacional aceitável e são comumente utilizados para problemas de grande porte. Existem duas famílias de métodos: heurísticas construtivas e meta-heurísticas. As heurísticas construtivas são projetadas para resolver um problema específico. Já as meta-heurísticas são de propósito geral, podendo assim ser aplicadas para resolver diferentes problemas de otimização. Existem na literatura diferentes meta-heurísticas tais como: ILS (*Iterative Local Search*), IG (*Iterated Greedy*), GRASP (*Greedy Randomized Adaptive Search*), SA (*Simulated Annealing*), TS (*Tabu Search*), VNS (*Variable Neighborhood Search*), GA (*Genetic Algorithms*), ACO (*Ant Colony Optimization*), entre outros.

Nas próximas seções serão descritas as ideias principais dos métodos de resolução utilizados para resolver o problema abordado neste trabalho.

3.1 Proximity Search

O *Proximity Search* (PS) é um método proposto por Fischetti & Monaci (2014) para resolver problemas de programação linear inteira mista 0-1 (PLIM). Este método utiliza uma estratégia de resolução cujo objetivo é, a partir de uma solução inicial viável, encontrar uma sequência de soluções melhoradas para um dado modelo de PLIM do problema. O funcionamento do PS consiste em substituir a função objetivo do modelo por uma função de proximidade e utilizar a resolução deste novo modelo como heurística de refinamento. Este modelo é resolvido como uma espécie de caixa-preta, ou seja, os detalhes das técnicas utilizadas para resolvê-lo são abstraídos pela utilização de um software de otimização específico para resolução de modelos de PLIM. A seguir é apresentada uma formulação genérica de um modelo de PLIM:

$$min\ f(x) \tag{3.1}$$

$$g(x) \leq 0 \tag{3.2}$$

$$x_j \in \{0,1\}, \forall j \in J \tag{3.3}$$

Uma solução exata para um problema de PLIM é geralmente obtida usando algum método enumerativo, como por exemplo o *branch-and-bound*. A estratégia utilizada pelos métodos enumerativos para explorar os nós de uma árvore de busca (ramificação) se mostra eficaz na melhoria do limitante inferior (*lower bound*) entre os nós visitados da árvore e, assim, talvez prove a otimalidade da solução incumbente. No entanto, não possui uma boa eficiência em encontrar soluções viáveis de boa qualidade no início da busca, podendo não ser a melhor opção de resolução para problemas de grande porte. O PS tem como propósito suprir essa deficiência dos métodos enumerativos.

Para iniciar a busca, o PS necessita de uma solução viável x' que pode ser obtida através de uma heurística. Em seguida é adicionada uma restrição de corte ao conjunto de restrições do modelo. Essa restrição de corte é descrita como:

$$f(x) \leq f(x') - \theta \tag{3.4}$$

onde $\theta > 0$ é chamada de variável de corte. Essa restrição garante que somente soluções melhores que a solução inicial (ou solução corrente) possam ser encontradas através da resolução do modelo. Isso faz com que o espaço de busca fique mais restrito.

Após a inserção desta restrição, a função objetivo $f(x)$ do modelo é modificada

a fim de guiar heuristicamente a busca e aumentar as chances de encontrar soluções viáveis melhores que x' na parte inicial da busca. A estratégia utilizada pelo PS é substituir $f(x)$ por uma função de proximidade $\Delta(x, x')$ que penalize a solução x de acordo com a sua distância em relação a x'. A distância utilizada pelo PS é a distância de Hamming apresentada a seguir:

$$\Delta(x, x') := \sum_{j \in J: x'_j = 0} x_j + \sum_{j \in J: x'_j = 1} (1 - x_j) \tag{3.5}$$

onde $x = (x_1, ..., x_n)$, $x' = (x'_1, ..., x'_n)$ e J é o conjunto de índices. Ou seja, a distância de Hamming corresponde ao número de variáveis binárias diferentes das soluções x e x'.

Na Figura 3.1 é apresentado um exemplo do cálculo da distância de Hamming entre duas soluções x e x' com variáveis binárias. Como pode ser visto na figura, $\Delta(x, x')$ é igual a 4, isso significa que existem 4 variáveis com valores diferentes. Quanto maior é este valor, maior será a distância entre as duas soluções.

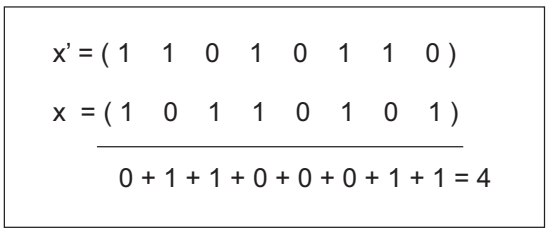


Figura 3.1. Exemplo da distância de Hamming entre duas variáveis x e x'

Com a aplicação da restrição de corte e da função de proximidade, o modelo tem como objetivo encontrar uma solução viável com valor melhor que $f(x') - \theta$ e que a solução x seja o mais semelhante (próxima) possível de x'.

Para melhorar o entendimento, o funcionamento do PS é descrito pelo pseudocódigo do Algoritmo 1 (Fischetti & Monaci, 2014). Na linha 2, uma solução inicial viável x' é gerada. A solução inicial pode ser encontrada utilizando uma heurística construtiva rápida ou resolvendo o modelo original do problema. A resolução do modelo deve estar restrita a retornar a primeira solução viável encontrada, ou limitada moderadamente por um tempo de execução ou por uma certa quantidade de nós explorados.

Na linha 4, uma restrição de corte é adicionada ao modelo utilizando um determinado valor para a variável θ. Ao utilizar um valor muito pequeno para θ, por exemplo $\theta = 1$, é possível que se tenha ao longo das iterações do PS uma série de modelos fáceis de resolver, onde cada um produz uma pequena melhora em relação

a solução corrente x'. Aplicar uma estratégia para definir um melhor valor para θ pode levar a uma melhora de desempenho do método.

Na linha 5, a função objetivo do modelo é substituída pela função de proximidade $\Delta(x, x')$, que é definida de acordo com a distância de Hamming. Em seguida, na linha 6, um software específico é chamado para resolver o novo modelo criado. O objetivo é encontrar uma nova solução x^* melhor que a solução corrente x' e com menor distância entre elas.

Na linha 7 é verificado se foi possível encontrar uma solução viável x^* com a resolução do modelo. Se verdadeiro, a função objetivo $f(x^*)$ do modelo original é calculada. Esse modelo original se refere ao modelo do problema antes da função objetivo ser substituída pela função de proximidade. Em seguida, na linha 9, a função de proximidade $\Delta(x, x')$ deve ser atualizada, fazendo a solução corrente x' ser igual a solução x^* encontrada pelo modelo. O valor da variável de corte θ pode ser atualizado aplicando uma estratégia específica para isso ou o seu valor pode ser mantido fixo.

O PS executa essas etapas apresentadas de forma iterativa até que um critério de parada seja satisfeito, conforme definido na linha 3. Esse critério pode ser definido por um período de tempo ou o PS pode ser finalizado quando a resolução do modelo provar que não existem soluções viáveis para uma determinada restrição de corte.

Algoritmo 1: Proximity Search

```
   saída: um solução viável x';
 1 início
 2 |  x' := solução inicial viável gerada por uma heurística;
 3 |  enquanto critério de parada não satisfeito faça
 4 |  |  Adicionar a restrição de corte f(x) ≤ f(x') − θ ao modelo;
 5 |  |  Substituir a função objetivo f(x) do modelo pela função de
   |  |  proximidade Δ(x, x');
 6 |  |  x* := a melhor solução obtida ao resolver o novo modelo;
 7 |  |  se x* <> ∅ então
 8 |  |  |  x* := argmin{f(x) : g(x) ≤ 0, x_j = x*_j ∀j ∈ J}
 9 |  |  Atualizar Δ(x, x') fazendo x' := x* e/ou atualizar θ;
10 |  fim
11 fim
```

3.2 Heurísticas

Diferentemente dos métodos exatos, as heurísticas são métodos aproximados de geração de soluções de alta qualidade utilizando um tempo computacional aceitável, mas elas não garantem encontrar soluções ótimas e nem provar sua otimalidade. A seguir são apresentadas as definições e as principais características das heurísticas construtivas, das heurísticas de melhoramento e das meta-heurísticas.

3.2.1 Heurísticas Construtivas

A heurística construtiva é uma técnica muito conhecida e utilizada por ser simples de ser desenvolvida e, geralmente, possui baixa complexidade computacional (Talbi, 2009). Além disso, a heurística construtiva é comumente utilizada para construir uma solução inicial viável de boa qualidade para heurísticas de melhoramento e meta-heurísticas, que serão discutidas nas próximas seções. Segundo Arroyo & Armentano (2004), o desempenho das meta-heurísticas também podem depender da qualidade da solução inicial gerada pela heurística construtiva.

No pseudocódigo do Algoritmo 2 é exemplificado o funcionamento básico de uma heurística construtiva. Primeiramente, a heurística começa com uma solução vazia ($x = \emptyset$) e um conjunto de elementos $E = \{e_1, e_2, ..., e_n\}$. A partir de um critério de seleção, em cada iteração i da heurística é selecionado um elemento distinto $e^* \in E$, que, em seguida, é atribuído à solução ($x \leftarrow e^*$). A heurística finaliza quando a solução é completamente construída, ou seja, até que não existam mais elementos a serem inseridos ($E = \emptyset$).

Algoritmo 2: Heurística Construtiva

```
Saída: uma solução viável x;
1 início
2 |  x := ∅;
3 |  enquanto |E| ≥ 0 faça
4 |  |  e* := Heurística-Local(E); // Selecionar um elemento ∈ E
5 |  |  x := x ∪ e*;
6 |  |  E := E − {e*};
7 |  fim
8 fim
```

3.2.2 Heurísticas de Melhoramento

Heurística de melhoramento, também conhecida como busca local, é um método usado para melhorar uma determinada solução aplicando movimentos nos elementos da solução. Os movimentos são modificações realizadas na solução a fim de gerar uma nova solução, no entanto, mantendo boa parte das características da solução original. Essa nova solução é denominada de solução vizinha. As soluções vizinhas geradas a partir de um grupo de movimentos formam a vizinhança da solução.

Em cada iteração de uma busca local, a solução atual x é substituída por uma solução vizinha x' se $f(x') < f(x)$. A busca termina quando todos as soluções vizinhas geradas são piores que a solução atual, representando que uma solução ótima local foi encontrada.

As buscas locais podem ter diferentes estratégias de seleção da melhor solução vizinha. A seguir são apresentadas duas principais estratégias:

- **Melhor aprimorante ou *Best Improvement***: também conhecido como *método da descida* para problemas de minimização ou *método da subida*, para problema de maximização, nesta estratégia é selecionado o vizinho com melhor valor de função objetivo. Portanto, todos os movimentos são avaliados para seleção do melhor vizinho. Devido a sua exploração exaustiva, este tipo de abordagem pode ter um gasto computacional para grandes vizinhanças.

- **Primeiro aprimorante ou *First Improvement***: nesta estratégia seleciona-se o primeiro vizinho que melhore a solução corrente. Encontrando um vizinho melhor, a geração da vizinhança é interrompida e o melhor vizinho assume o lugar da solução atual. Esta abordagem evita a exploração exaustiva, sendo que somente no pior caso toda a vizinhança é explorada. No entanto, essa estratégia fica limitada ao primeiro ótimo local.

No Algoritmo 3 é apresentado um pseudocódigo demonstrando o funcionamento de uma busca local com critério de seleção baseado no melhor aprimorante. A partir de uma solução inicial x_0, um conjunto N é criado contendo as soluções vizinhas geradas e em seguida a melhor solução vizinha x' é obtida. Ocorre uma melhoria quando $f(x')$ é melhor que a solução corrente $f(x)$. Neste caso, a solução x é substituída por x' e a busca local se repete a partir desta nova solução. A busca termina quando o critério de parada é satisfeito ou quando nenhuma solução vizinha melhorar a solução corrente.

Algoritmo 3: Busca Local (x_0)

```
   saída: a solução x encontrada;
 1 início
 2 |  x := x_0; // Solução Inicial
 3 |  enquanto Critério de parada não satisfeito faça
 4 |  |  N := GerarVizinhanca(x)
 5 |  |  x' := MelhorVizinho(N); // Melhor aprimorante
 6 |  |  se f(x') ≤ f(x) então
 7 |  |  |  x = x'; // Atualizar solução corrente
 8 |  |  senão
 9 |  |  |  Pare; // Não há mais vizinhos melhores
10 |  |  fim
11 |  fim
12 fim
```

3.2.3 Meta-heurísticas

Para problemas onde não são conhecidos algoritmos exatos que resolva o problema em tempo aceitável, o uso de meta-heurísticas é fortemente recomendado. A principal vantagem das meta-heurísticas é ser capaz de fornecer soluções de alta qualidade em um tempo razoavelmente restrito e também para problemas de grande porte. No entanto, diferentemente dos métodos exatos, as meta-heurísticas não garantem encontrar soluções ótimas e não definem o quanto a solução está próxima da solução ótima. As meta-heurísticas podem ser definidas como procedimentos de propósito geral usados para resolver diferentes problemas de otimização combinatória. A estrutura de uma meta-heurística deve ser adaptada ao problema que se deseja resolver.

A seguir é realizada uma revisão das meta-heurísticas utilizadas para resolver o problema proposto neste trabalho.

3.2.3.1 Greedy Randomized Adaptive Search

Inicialmente proposta por Feo & Resende (1995), a meta-heurística *Greedy Randomized Adaptive Search* (GRASP) é um procedimento iterativo que combina uma fase de construção, onde, iterativamente, uma solução é construída inserindo um elemento por vez, e uma fase de melhoramento, onde uma busca local é aplicada a fim de identificar um ótimo local. O procedimento termina quando um critério de parada é satisfeito, e então a melhor solução encontrada é retornada.

No Algoritmo 4 é apresentado um pseudocódigo que descreve o funcionamento básico do GRASP. Na linha 4 é aplicada uma fase de construção. Nesta fase, uma

solução viável x é obtida realizando os seguintes passos:

- Determinar uma lista de elementos candidatos e avaliar o benefício de incluir cada elemento na solução utilizando uma função de avaliação;
- Criar uma lista restrita de candidatos com os elementos melhor avaliados, em seguida um elemento desta lista é selecionado aleatoriamente e inserido na solução;
- Na próxima iteração, a função de avaliação de cada elemento é adaptada considerando o novo estado da solução.

Após o término da fase construtiva, uma busca local é aplicada na solução x a fim de obter uma solução melhorada x', conforme linha 5. Após a busca local, a solução x^* será substituída por x' se $f(x') < f(x^*)$. Se o critério de parada não for satisfeito, o algoritmo repete as fases de construção e busca local obtendo uma nova solução. Caso contrário, o GRASP finaliza e retorna a melhor solução armazenada.

Algoritmo 4: GRASP

```
   saída: a melhor solução encontrada (x*);
1  início
2  |  x* := ∞;
3  |  enquanto Critério de parada não satisfeito faça
4  |  |  x := Construtivo();
5  |  |  x' := BuscaLocal(x);
6  |  |  se f(x') < f(x*) então
7  |  |  |  x* := x';
8  |  fim
9  fim
```

3.2.3.2 Iterated Greedy

A meta-heurística Iterated Greedy (IG) foi introduzida por Ruiz & Stützle (2007) para o problema de *flowshop scheduling*. O IG é uma meta-heurística simples e fácil de ser usada, podendo ser escrita em poucas linhas de código.

O pseudocódigo do IG é apresentado pelo Algoritmo 5. Como pode ser observado na linha 2, uma solução inicial x_0 é gerada, podendo ser obtida usando uma heurística construtiva, e, em seguida, na linha 3, uma busca local é aplicada a fim de refinar a solução inicial.

Nas linhas de 4 a 13, o IG tenta melhorar iterativamente a solução atual x através de duas etapas. A primeira etapa, descrita na linha 5, é chamada de *Destruição-Construção*. Esta etapa consiste em destruir a solução atual removendo uma certa quantidade d de seus elementos e, logo após, a solução é reconstruída baseando-se em uma heurística construtiva gulosa (ou seja, os elementos removidos são reinseridos de forma gulosa na solução).

A segunda etapa consiste em aplicar uma busca local sobre a solução reconstruída x' a fim de encontrar um novo ótimo local x''. Nas linhas de 7 a 12, a solução corrente x e a melhor solução encontrada x^* são atualizadas. Caso a solução x'' não melhore a solução atual x, um critério de aceitação é aplicado, conforme linha 12. Esse critério define se o ótimo local em x'' será aceito, mesmo havendo uma piora em relação a x. O objetivo de criar um critério de aceitação é adotar um balanceamento entre intensificação e diversificação da solução. Essas etapas descritas são executadas até que um critério de parada seja satisfeito e então a melhor solução encontrada é retornada.

Algoritmo 5: IG

```
   saída: a melhor solução encontrada (x*);
 1 início
 2 |  x_0 := Construtivo();
 3 |  x := BuscaLocal(x_0);
 4 |  enquanto Critério de parada não satisfeito faça
 5 |  |  x' := Destruição-Construição(x, d);
 6 |  |  x'' := BuscaLocal(x');
 7 |  |  se f(x'') < f(x) então
 8 |  |  |  x := x'';
 9 |  |  |  se f(x) ≤ f(x*) então
10 |  |  |  |  x* := x;
11 |  |  senão
12 |  |  |  x := CritérioAceitação(x, x'', histórico);
13 |  fim
14 fim
```

Capítulo 4

Modelagens Matemáticas para o Problema $R_m \mid prec, S_{ijk} \mid C_{max}$

Neste capítulo são apresentados três modelos de programação linear inteira mista (PLIM) para o problema $R_m \mid Prec, S_{ijk} \mid C_{max}$. Os modelos propostos são baseados em modelos de problemas similares encontrados na literatura, sendo adaptados a fim de atender às restrições de precedência e tempos de preparação. Em cada uma das seções a seguir são apresentados os parâmetros de entrada, as variáveis de decisão e o modelo matemático.

4.1 Modelo 1

A seguir é apresentado um modelo de programação linear inteira mista (PLIM) adaptado de Rabadi et al. (2006) e baseado no modelo inicialmente proposto por Manne (1960). Essa adaptação consiste em acrescentar as restrições de precedência ao modelo. O modelo, denotado por Modelo 1, utiliza os seguintes parâmetros:

- $J = \{1, ..., n\}$: conjunto de tarefas, sendo n o número de tarefas;
- $M = \{1, ..., m\}$: conjunto de máquinas, sendo m o número de máquinas;
- $J_0 = J \cup \{0\}$: conjunto de tarefas J acrescido da tarefa fictícia 0. A tarefa $j = 0$ é sequenciada pelo modelo antes da primeira tarefa e depois da última tarefa em cada máquina i. Seu objetivo é sinalizar o início e o fim de cada sequenciamento para as restrições do modelo.
- $A = \{a_1, ..., a_i, ..., a_p\}$: conjunto das p restrições de precedência, onde a i-ésima restrição $a_i = (j, k)$ representa que a tarefa j deve preceder a tarefa k;

- B: um número inteiro suficientemente grande, definido pelo somatório do maior tempo de processamento ($max(p_{ij})$) e do maior tempo de preparação ($max(S_{ikj})$) de cada tarefa j;
- p_{ij}: tempo de processamento da tarefa j na máquina i;
- S_{ijk}: tempo de preparação ou *setup time* necessário para processar a tarefa k sequenciada logo após a tarefa j na máquina i.

O modelo utiliza as seguintes variáveis de decisão:

- x_{ijk}: igual a 1 se a tarefa k é processada após a tarefa j na máquina i e 0 caso contrário;
- C_j: tempo de conclusão da tarefa j;
- C_{max}: tempo máximo de conclusão das tarefas ou *makespan*.

A formulação do modelo é apresentada a seguir:

$$Min \quad C_{max} \tag{4.1}$$

$$s.a:$$

$$\sum_{j=0,j\neq k}^{n}\sum_{i=1}^{m} x_{ijk} = 1, \quad \forall k \in J \tag{4.2}$$

$$\sum_{j=0,j\neq h}^{n} x_{ijh} = \sum_{k=0,k\neq h}^{n} x_{ihk}, \quad \forall h \in J,\ \forall i \in M \tag{4.3}$$

$$\sum_{k=0}^{n} x_{i0k} \leq 1, \quad \forall i \in M \tag{4.4}$$

$$C_j + \sum_{i=1}^{m}(S_{ijk} + p_{ik})x_{ijk} + B(\sum_{i=1}^{m} x_{ijk} - 1) \leq C_k, \forall j \in J_0,\ \forall k \in J \tag{4.5}$$

$$C_k - C_j \geq \sum_{i=1}^{m}\sum_{l=0}^{n} p_{ik} \cdot x_{ilk}, \qquad \forall (j,k) \in A \tag{4.6}$$

$$C_0 = 0 \tag{4.7}$$

$$C_{max} \geq C_j, \quad \forall j \in J \tag{4.8}$$

$$C_j \geq 0, \quad \forall j \in J \tag{4.9}$$

$$x_{ijk} \in \{0,1\}, \quad \forall j,k \in J_0,\ \forall i \in M. \tag{4.10}$$

A Equação (4.1) representa a função objetivo que é a minimização do *makespan*. As restrições (4.2) garantem que cada tarefa será processada apenas uma vez

e por uma única máquina. As restrições (4.3) garantem que cada tarefa deve ter somente uma tarefa imediatamente predecessora e uma tarefa imediatamente sucessora. As restrições (4.4) fazem com que no máximo uma tarefa seja sequenciada após a tarefa fictícia 0, ou seja, no máximo uma tarefa pode ser a primeira tarefa de uma máquina. Quando não existir nenhuma tarefa sucessora a tarefa 0 em uma determinada máquina, significa que a máquina não é utilizada para processar nenhuma tarefa. As restrições (4.5) são usadas para calcular o tempo de conclusão de cada tarefa e também não permitem que uma tarefa seja predecessora e sucessora de uma mesma tarefa. As restrições (4.6) representam as restrições de precedência. Essas restrições garantem que cada tarefa não pode ser inicializada antes que todas as suas tarefas predecessoras tenham sido finalizadas. A restrição (4.7) obriga a tarefa fictícia 0 ter tempo de conclusão igual a zero, portanto, não interfere no cálculo do *makespan*. As restrições (4.8) definem o *makespan* (C_{max}) como sendo o maior tempo de conclusão das tarefas. Finalmente, as restrições (4.9) e (4.10), respectivamente, garantem que o tempo de conclusão das tarefas tenha valor positivo e define que as variáveis x_{ijk} sejam binárias.

4.2 Modelo 2

O modelo matemático apresentado a seguir é uma adaptação do modelo proposto por Vallada & Ruiz (2011) que originalmente foi utilizado para resolver o problema $R_m \mid S_{ijk} \mid C_{max}$. O modelo, denotado por Modelo 2, utiliza os mesmos parâmetros e variáveis utilizados pelo Modelo 1. A seguir, eles são novamente descritos a fim de facilitar a compreensão do modelo.

- $J = \{1, ..., n\}$: conjunto de tarefas, sendo n o número de tarefas;
- $M = \{1, ..., m\}$: conjunto de máquinas, sendo m o número de máquinas;
- $J_0 = J \cup \{0\}$: conjunto de tarefas J acrescido da tarefa fictícia 0. A tarefa $j = 0$ é sequenciada pelo modelo antes da primeira tarefa e depois da última tarefa em cada máquina i. Seu objetivo é sinalizar o início e o fim de cada sequenciamento para as restrições do modelo.
- $A = \{a_1, ..., a_i, ..., a_p\}$: conjunto das p restrições de precedência, onde a i-ésima restrição $a_i = (j, k)$ representa que a tarefa j deve preceder a tarefa k;

- B: um número inteiro suficientemente grande, definido pelo somatório do maior tempo de processamento ($max(p_{ij})$) e do maior tempo de preparação ($max(S_{ikj})$) de cada tarefa j;
- p_{ij}: tempo de processamento da tarefa j na máquina i;
- S_{ijk}: tempo de preparação ou *setup time* necessário para processar a tarefa k sequenciada logo após a tarefa j na máquina i.

O modelo utiliza as seguintes variáveis de decisão:

- x_{ijk}: igual a 1 se a tarefa k é processada após a tarefa j na máquina i e 0 caso contrário;
- C_j: tempo de conclusão da tarefa j;
- C_{max}: tempo máximo de conclusão das tarefas ou *makespan*.

A formulação do modelo é apresentada a seguir:

$$Min \quad C_{max} \tag{4.11}$$

$$s.a:$$

$$\sum_{i=1}^{m} \sum_{j=0;j\neq k}^{n} x_{ijk} = 1, \quad \forall k \in J \tag{4.12}$$

$$\sum_{i=1}^{m} \sum_{k=1;j\neq k}^{n} x_{ijk} \leq 1, \quad \forall j \in J \tag{4.13}$$

$$\sum_{k=1}^{n} x_{i0k} \leq 1, \quad \forall i \in M \tag{4.14}$$

$$\sum_{h=0;h\neq k;h\neq j}^{n} x_{ihj} \geq x_{ijk}, \quad \forall j,k \in J,\, j \neq k, \forall i \in M \tag{4.15}$$

$$C_k + B \cdot (1 - x_{ijk}) \geq C_j + S_{ijk} + p_{ik}, \quad \forall j \in J_0, \forall k \in J \tag{4.16}$$

$$C_j - C_k \geq \sum_{i=1}^{m} \sum_{l=0}^{n} p_{ij} \cdot x_{ilj}, \quad \forall (j,k) \in A \tag{4.17}$$

$$C_0 = 0 \tag{4.18}$$

$$C_{max} \geq C_j, \quad \forall j \in J \tag{4.19}$$

$$C_j \geq 0, \quad \forall j \in J \tag{4.20}$$

$$x_{ijk} \in \{0,1\}, \quad \forall j,k \in J_0,\, \forall i \in M. \tag{4.21}$$

A função objetivo (4.11) é minimizar o *makespan*. As restrições (4.12) garantem que deve existir uma única tarefa j predecessora imediata a uma tarefa k em

uma máquina i. As restrições (4.13) garantem que o número máximo de tarefas sucessoras imediatas de uma dada tarefa j seja igual a 1. As restrições (4.14) garantem que deve existir no máximo uma tarefa sucessora a tarefa fictícia 0. Quando não existir nenhuma tarefa sucessora a $j = 0$ em uma determinada máquina, significa que a máquina não é utilizada para processar nenhuma tarefa. As restrições (4.15) garantem que, em uma mesma máquina, se uma tarefa j é predecessora imediata de uma tarefa k, obrigatoriamente deve existir uma tarefa imediatamente predecessora de j. As restrições (4.16) são responsáveis por limitar os tempos de conclusão de cada tarefa. Se uma tarefa k é sequenciada logo após a tarefa j na máquina i, $x_{ijk} = 1$, então, o tempo de conclusão da tarefa k deve ser maior ou igual ao tempo de conclusão da tarefa j somado ao tempo de preparação S_{ijk} mais o tempo de processamento p_{ik}. As restrições (4.17) representam as restrições de precedência, onde cada tarefa não pode ser inicializada antes que todas as suas tarefas predecessoras tenham sido finalizadas. A restrição (4.18) obriga que a tarefa fictícia 0 tenha tempo de conclusão igual a zero. As restrições (4.19) definem o *makespan* (C_{max}) como sendo o tempo máximo de conclusão das tarefas. Finalmente, as restrições (4.20) e (4.21), respectivamente, garantem que o tempo de conclusão das tarefas tenha valor positivo e define que as variáveis x_{ijk} sejam binárias.

4.3 Modelo 3

O modelo matemático apresentado a seguir, denotado por Modelo 3, é uma adaptação do modelo proposto por Liu (2013) utilizado para resolver o problema $R_m \mid Prec \mid T$ e baseado no trabalho de Wagner (1959). Este modelo se difere dos modelos apresentados anteriormente pelo fato de considerar as posições com que as tarefas são alocadas nas máquinas. Os seguintes parâmetros são utilizadas por este modelo:

- $J = \{1, ..., n\}$: conjunto de tarefas, sendo n o número de tarefas;
- $M = \{1, ..., m\}$: conjunto de máquinas, sendo m o número de máquinas;
- $A = \{a_1, ..., a_i, ..., a_p\}$: conjunto das p restrições de precedência, onde a i-ésima restrição $a_i = (j, k)$ representa que a tarefa j deve preceder a tarefa k;
- B: um número inteiro suficientemente grande, definido pelo somatório do maior tempo de processamento ($max(p_{ij})$) e do maior tempo de preparação ($max(S_{ikj})$) de cada tarefa j;

- p_{ij}: tempo de processamento da tarefa j na máquina i;
- S_{ijk}: tempo de preparação ou *setup time* necessário para processar a tarefa k sequenciada logo após a tarefa j na máquina i.
- ϕ: número máximo de posições em cada máquina onde as tarefas são sequenciadas. Cada máquina pode ter no máximo n posições.

O modelo utiliza as seguintes variáveis de decisão:

- x_{ijr}: igual a 1 se a tarefa j está sequenciada na posição r da máquina i, caso contrário, igual a zero;
- σ_{itkr}: igual a 1, se a tarefa k é processada logo após a tarefa t na posição r da máquina i, caso contrário, igual a zero;
- C_j: tempo de conclusão da tarefa j;
- C_{max}: tempo máximo de conclusão das tarefas ou *makespan*.

A formulação do modelo é apresentada a seguir:

$$Min \quad C_{max} \tag{4.22}$$

$$st.$$

$$\sum_{i=1}^{m}\sum_{r=1}^{\phi} x_{ijr} = 1, \quad \forall j \in J \tag{4.23}$$

$$\sum_{j=1}^{n} x_{ijr} \leq 1, \quad \forall r \in \{1,...,\phi\}, \forall i \in M \tag{4.24}$$

$$\sum_{k=1}^{n} x_{ikr} - \sum_{j=1}^{n} x_{ij(r-1)} \leq 0, \quad \forall i \in M, \forall r \in \{2,...,\phi\} \tag{4.25}$$

$$C_k \geq (S_{i0k} + p_{ik})\, x_{ik1}, \quad \forall k \in J, \forall i \in M \tag{4.26}$$

$$C_k - C_j + B\,(2 - x_{ikr} - x_{ij(r-1)}) \geq S_{ijk} + p_{ik}, \tag{4.27}$$
$$\forall j,k \in J, j \neq k, \forall i \in M, \forall r \in \{2,...,\phi\}$$

$$C_k - C_j \geq \sum_{i=1}^{m} p_{ik} \cdot x_{ik1}, \quad \forall (j,k) \in A \tag{4.28}$$

$$C_k - C_j \geq \sum_{t=1}^{n}\sum_{i=1}^{m}\sum_{r=2}^{\phi} p_{ik} \cdot (x_{it(r-1)} \cdot x_{ikr}), \quad \forall (j,k) \in A \tag{4.29}$$

$$C_{max} \geq C_j, \quad \forall j \in J \tag{4.30}$$

$$C_j \geq 0, \quad \forall j \in J \tag{4.31}$$

$$x_{ijr} \in \{0,1\}, \quad \forall i \in M,\ \forall j \in J\ \forall r \in \{1,...,\phi\} \tag{4.32}$$

Como pode ser observado no modelo, as restrições (4.18) são não-lineares por causa da multiplicação entre as variáveis de decisão $x_{it(r-1)}$ e x_{ikr}. Para linearizar o modelo é realizada a seguinte substituição:

- $\sigma_{itkr} = x_{it(r-1)} \cdot x_{ikr}$

Portanto, as restrições (4.29) serão substituídas pelas restrições (4.33) e são adicionadas as restrições (4.34), (4.35) e (4.36) necessárias para a linearização do modelo:

$$C_k - C_j \geq \sum_{t=1}^{n}\sum_{i=1}^{m}\sum_{r=2}^{\phi}(S_{itk} + p_{ik}) \cdot \sigma_{itkr}, \quad \forall(j,k) \in A \tag{4.33}$$

$$-x_{it(r-1)} + \sigma_{itkr} \leq 0, \tag{4.34}$$

$$-x_{ikr} + \sigma_{itkr} \leq 0, \tag{4.35}$$

$$x_{it(r-1)} + x_{ikr} - \sigma_{itkr} \leq 1, \tag{4.36}$$

$$\forall k,t \in J, i \neq t, \forall i \in M, \ \forall r \in \{2,...,\phi\}$$

A Equação (4.22) representa a minimização do *makespan*. As restrições (4.23) garantem que cada tarefa j deve ser designada para apenas uma única posição r de uma única máquina i. As restrições (4.24) garantem que no máximo uma tarefa será designada para cada posição. As restrições em (4.25) garantem que cada tarefa, partindo da posição $r = 2$, deve ter uma tarefa na posição anterior $(r - 1)$. As restrições em (4.26) e (4.27) garantem que o tempo de conclusão de uma tarefa k em uma máquina i seja maior ou igual ao tempo de conclusão da tarefa antecessora j mais o tempo de processamento e tempo de preparação da tarefa k. As restrições (4.28) e (4.29) obedecem as relações de precedência entre as tarefas. As restrições (4.30) garantem que o *makespan* seja o tempo máximo de conclusão das tarefas. As restrições de (4.31) e (4.32) definem os domínios das variáveis. E, por fim, as restrições de (4.33-4.36) são responsáveis pela linearização necessária provocada pela restrição (4.29).

4.4 Geração de Instâncias para o Problema

Devido ao fato de não existirem trabalhos na literatura que consideram as mesmas características do problema abordado neste trabalho, foram geradas diferentes instâncias para o problema. Essas instâncias possuem diferentes tamanhos e são utilizadas pelos experimentos computacionais realizados neste trabalho.

Foram gerados três grupos de instâncias que variam de acordo com o número de tarefas (n), o número de máquinas (m), o intervalo de tempo de preparação (s) e o nível de densidade das restrições de precedência (d). O primeiro grupo é formado por instâncias pequenas com $n \in \{6, 8, 10, 12\}$ e $m \in \{2, 3, 4, 5\}$. O segundo conjunto é formado por instâncias de tamanho médio com $n \in \{15, 20, 25, 30\}$ e $m \in \{2, 5, 10\}$. E o terceiro grupo é formado por instâncias grandes com $n \in \{30, 60, 90, 120, 150, 180\}$ e $m \in \{5, 10, 15\}$. Em todos os três grupos, as instâncias foram geradas usando $d \in \{0.2, 0.4, 0.6\}$ e intervalos de tempo de preparação $s \in \{[0,9], [0,49], [0,99]\}$. Foram geradas 5 instâncias para cada combinação de n, m, d e s. No total, foram geradas 720 instâncias pequenas, 540 intermediárias e 810 grandes.

A geração das instâncias é similar ao apresentado por Liu (2013). O tempo de processamento de cada tarefa é um número inteiro gerado pela distribuição uniforme dentro do intervalo de [1,100]. As restrições de precedência são geradas usando a probabilidade P_{ij} de uma tarefa i ter relação de precedência com a tarefa j. Para uma dada densidade d, a probabilidade é definida por:

$$P_{ij} = \frac{d \times (1-d)^{j-i-1}}{(1-d) \times (1-(1-d)^{j-i-1})}, \forall\ 1 \leq i < j \leq n, i \neq j. \tag{4.37}$$

4.5 Experimentos Computacionais com os Modelos PLIM

Nesta seção são apresentados os experimentos computacionais realizados utilizando as instâncias pequenas e médias a fim de definir, entre os três modelos propostos, o modelo com melhor desempenho. Os modelos matemáticos foram implementados utilizando a biblioteca ILOG Concert. Com essa biblioteca foi possível implementar os modelos usando a linguagem C++ e resolvê-los através do software IBM ILOG CPLEX 12.6.2. Os experimentos foram realizados em um computador com processador Intel Core i7 com 4.00 GHz de clock, 32GB de memória RAM e sistema operacional Windows 8 64 bits. Em todos os experimentos, os modelos foram limitados a 3.600 segundos de tempo de CPU para cada instância.

Para avaliar os modelos foram utilizadas as seguintes métricas: a média do *makespan* (C_{max}), a média do Gap (%) e a média do tempo de CPU em segundos. Nos experimentos, o Gap representa a distância relativa entre o valor da função objetivo da melhor solução encontrada (C_{max}) e o valor do *lower bound* obtido pelo modelo.

4.5.1 Experimento 1: Comparação entre os Modelos utilizando Instâncias Pequenas

Neste experimento foram executados o Modelo 1, o Modelo 2 e o Modelo 3 utilizando o conjunto de instâncias pequenas. No total, foram testadas 720 instâncias, onde o Modelo 1 e Modelo 2 conseguiram obter soluções ótimas para todas as instâncias. No entanto, o Modelo 3 obteve 646 soluções ótimas e 74 soluções viáveis. Este experimento ficou limitado ao conjunto de instâncias pequenas, pois o Modelo 3 consegue encontrar soluções viáveis dentro do tempo estipulado somente para instâncias com até 12 tarefas.

Na Tabela 4.1 são apresentados os resultados obtidos para os três modelos. Como pode ser observado na tabela, o Modelo 1 e o Modelo 2 obtiveram média total de *makespan* igual a 178,66 e o Modelo 3 obteve média total igual a 179,30. Além disso, o Modelo 1 e Modelo 2 conseguiram Gap igual a 0% para todas as instâncias. Isso significa que todas as soluções obtidas pelos dois modelos são soluções ótimas. Já o Modelo 3 obteve Gap superior a 0% para instâncias com mais de 10 tarefas. Analisando a média de tempo de execução dos três modelos, o Modelo 1 obteve o menor tempo gasto para executar todas as instâncias, sendo o tempo médio igual a 1,49 segundos.

Com esses resultados é possível concluir que, para um conjunto de instâncias pequenas, o Modelo 1 possui o melhor desempenho e o Modelo 3 apresenta o pior desempenho à medida que as instâncias vão aumentando de tamanho.

Tabela 4.1. Comparativo entre os Modelos para Instâncias Pequenas.

		Modelo 1			Modelo 2			Modelo 3		
n	m	C_{max}	Gap(%)	Tempo(s)	C_{max}	Gap(%)	Tempo(s)	C_{max}	Gap(%)	Tempo(s)
6	2	**195,42**	**0,00**	**0,09**	**195,42**	**0,00**	0,10	**195,42**	**0,00**	0,34
6	3	**137,78**	**0,00**	**0,08**	**137,78**	**0,00**	0,10	**137,78**	**0,00**	0,46
6	4	**112,09**	**0,00**	**0,09**	**112,09**	**0,00**	0,10	**112,09**	**0,00**	0,55
6	5	**96,13**	**0,00**	**0,09**	**96,13**	**0,00**	0,11	**96,13**	**0,00**	0,61
8	2	**244,76**	**0,00**	**0,16**	**244,76**	**0,00**	0,18	**244,76**	**0,00**	6,59
8	3	**161,76**	**0,00**	**0,15**	**161,76**	**0,00**	0,18	**161,76**	**0,00**	5,63
8	4	**135,24**	**0,00**	**0,15**	**135,24**	**0,00**	0,18	**135,24**	**0,00**	4,50
8	5	**103,38**	**0,00**	**0,12**	**103,38**	**0,00**	0,16	**103,38**	**0,00**	4,10
10	2	**290,24**	**0,00**	**1,76**	**290,24**	**0,00**	2,44	**290,24**	**0,00**	119,39
10	3	**196,40**	**0,00**	**0,39**	**196,40**	**0,00**	0,43	**196,40**	**0,00**	62,70
10	4	**161,91**	**0,00**	**0,24**	**161,91**	**0,00**	0,35	**161,91**	**0,00**	54,57
10	5	**134,56**	**0,00**	**0,22**	**134,56**	**0,00**	0,33	**134,56**	**0,00**	39,58
12	2	**330,36**	**0,00**	**17,40**	**330,36**	**0,00**	30,33	338,02	41,18	2730,41
12	3	**227,80**	**0,00**	**1,50**	**227,80**	**0,00**	3,04	230,20	24,97	2594,26
12	4	**183,36**	**0,00**	**0,92**	**183,36**	**0,00**	1,57	183,58	14,15	1900,26
12	5	**147,33**	**0,00**	**0,52**	**147,33**	**0,00**	0,88	147,33	9,50	1060,23
Médias		**178,66**	**0,00**	**1,49**	**178,66**	**0,00**	2,53	179,30	5,61	536,51

4.5.2 Experimento 2: Comparação entre o Modelo 1 e o Modelo 2 utilizando Instâncias de Médio Porte

Neste experimento foram executados o Modelo 1 e o Modelo 2 para o conjunto de instâncias de médio porte. Este experimento ficou limitado a este conjunto de instâncias, pois nenhum dos modelos encontram soluções viáveis para instâncias com mais de 40 tarefas dentro do tempo estabelecido de 3600 segundos. Do total de 540 instâncias, o Modelo 1 conseguiu encontrar 359 soluções ótimas, enquanto o Modelo 2 conseguiu encontrar 316. Na Tabela 4.2 é apresentado o valor do Gap médio e o tempo médio gasto de cada modelo para cada grupo ($n \times m$) de instâncias. Como pode ser observado, o Modelo 1 obteve a menor média de Gap e o menor tempo médio de execução. Portanto, é possível considerar que o Modelo 1 obteve o melhor desempenho dentre os três modelos abordados neste capítulo.

Tabela 4.2. Médias de C_{max}, Gap e tempo de execução entre o Modelo 1 e o Modelo 2 usando instâncias de médio porte.

		Modelo 1			Modelo 2		
n	m	C_{max}	Gap	Tempo (s)	C_{max}	Gap	Tempo (s)
15	2	**408,73**	**5,24**	**853,46**	409,78	7,20	1192,58
15	5	**169,89**	**0,00**	**2,45**	**169,89**	**0,00**	6,17
15	10	**99,02**	**0,00**	**1,08**	**99,02**	**0,00**	4,30
20	2	**544,69**	**26,69**	**2744,57**	544,84	30,07	2838,91
20	5	**195,89**	**0,29**	**214,72**	196,93	3,14	878,17
20	10	**126,09**	**0,00**	**6,63**	**126,09**	**0,00**	94,90
25	2	**690,00**	**36,39**	**3122,13**	719,89	41,21	3280,78
25	5	**242,64**	**10,68**	**1689,19**	266,33	24,05	2671,10
25	10	**130,93**	**0,00**	**95,08**	131,18	0,79	868,64
30	2	**850,44**	**50,01**	**3528,48**	912,96	54,01	3552,68
30	5	**311,09**	**29,85**	**3122,53**	385,84	45,57	3450,14
30	10	**159,60**	**0,79**	**738,87**	181,04	15,71	2161,10
Médias		**327,42**	**13,33**	**1343,27**	345,32	18,48	1749,96

4.5.3 Experimento 3: Desempenho do Modelo 1 por Máquina, Densidade e Tempos de Preparação

Neste experimento é feita uma análise sobre as seguintes características do problema: quantidade de máquinas, nível de densidade das precedências e o intervalo de tempo de preparação. O objetivo é verificar o quanto essas características afetam no desempenho do Modelo 1. Neste experimento também é considerado o conjunto de instâncias de médio porte.

Na Tabela 4.3, é comparado o desempenho do modelo em relação ao número de máquinas. Como observado, a quantidade de soluções ótimas (# opt) aumentam, a

média do Gap e o tempo médio gasto diminuem a medida que o número de máquinas aumenta. Portanto, podemos afirmar que, para o Modelo 1, quanto maior o número de máquinas, menor é o esforço gasto pelo modelo para encontrar uma solução ótima.

Tabela 4.3. Desempenho do Modelo 1 por máquina.

m	# opt	gap (%)	tempo (s)
2	57	29,58	2562,16
5	124	10,20	1257,22
10	178	0,20	210,42

Na Tabela 4.4 são apresentados os resultados da resolução do modelo agrupados pelo nível de densidade das precedências. Como visto anteriormente, quanto maior o nível de densidade, maior é a chance de ocorrer precedência entre cada par de tarefas. De acordo com a tabela, quanto maior a densidade, maior o número de soluções ótimas, menor é o valor da média do Gap e do tempo médio gasto. Então, pode-se concluir que um número mais elevado de restrições de precedência pode diminuir o número de soluções viáveis (espaço de soluções), facilitando assim a resolução do modelo.

Tabela 4.4. Desempenho do Modelo 1 por densidade.

Densidade	# opt	gap (%)	tempo (s)
0,2	99	19,80	1744,59
0,4	119	13,20	1395,17
0,6	141	6,99	890,04

Finalmente, a Tabela 4.5 apresenta os resultados obtidos agrupados por intervalo de tempo de preparação. Nesta tabela também é possível ver que o intervalo de tempo de preparação definido para as instâncias do problema pode afetar no desempenho do modelo. Neste caso, quanto maior o intervalo do tempo de preparação, maior é o esforço do modelo para encontrar uma solução ótima.

Tabela 4.5. Desempenho do Modelo 1 por tempo de preparação.

Intervalo	# opt	gap (%)	tempo (s)
[0,9]	130	9,01	1063,47
[0,49]	118	13,79	1399,05
[0,99]	111	17,18	1567,28

4.6 Conclusão

Neste capítulo foram apresentados três modelos de programação linear inteira mista (PLIM) para resolver o problema proposto. Os modelos foram resolvidos pelo software CPLEX limitados a 3600 segundos de tempo de execução para cada instância.

A fim de avaliar o desempenho dos modelos, foram realizados experimentos computacionais utilizando o conjunto de instâncias de pequeno e médio porte. Como critérios de avaliação, foi utilizada a média da função objetivo (C_{max}), o Gap e o tempo gasto por cada modelo.

Para as instâncias pequenas, o Modelo 1 e o Modelo 2 obtiveram os melhores resultados, sendo que o Modelo 1 ganha no tempo médio de execução gasto. Além disso, os dois modelos encontram soluções ótimas para todas as instâncias. O Modelo 3 apresentou o pior desempenho, levando um tempo maior para executar todas as instâncias e não obtendo soluções ótimas para todas as instâncias. Além disso, o Modelo 3 não conseguiu encontrar soluções viáveis para instâncias com mais de 15 tarefas.

Para as instâncias de médio porte, foram comparados o Modelo 1 e o Modelo 2. O Modelo 1 conseguiu melhores resultados, conseguindo provar otimalidade em 359 instâncias contra 316 do Modelo 2 de um total de 540 instâncias. Além disso, obteve a menor média total para o Gap e a menor média total de tempo de execução. Portanto, nos próximos capítulos, o Modelo 1 será utilizado como base para implementação de novos métodos de resolução e também utilizado nos experimentos computacionais a fim de comparar o desempenho das heurísticas propostas neste trabalho.

Capítulo 5

Heurísticas Construtivas para o Problema $R_m \mid prec, S_{ijk} \mid C_{max}$

Como visto no Capítulo 3, uma heurística construtiva é uma técnica simples e de baixa complexidade computacional onde a solução obtida é muito utilizada como solução inicial para diferentes heurísticas de melhoramento e meta-heurísticas. Portanto, neste capítulo são abordadas sete heurísticas construtivas, cada uma baseada em uma determinada regra de prioridade, a fim de obter soluções de qualidade razoável para o problema abordado.

5.1 Heurística Construtiva Baseada em Regras de Prioridade

Nesta seção é apresentado o funcionamento da heurística construtiva (HC) baseada em regras de prioridade. Esta heurística foi inicialmente proposta por Liu & Yang (2011) para resolver o problema de sequenciamento em máquinas paralelas não-relacionadas considerando apenas restrição de precedência. A heurística utiliza, em cada iteração, uma determinada regra de prioridade para selecionar uma tarefa j e também uma máquina i onde a tarefa deve ser sequenciada. Após n iterações, todas as tarefas já foram sequenciadas e então uma solução viável é obtida. Portanto, a principal vantagem da heurística é gerar uma solução viável de boa qualidade e com baixa complexidade computacional.

A seguir são descritos os conjuntos e as variáveis utilizadas pela heurística:

J : conjunto de n tarefas $j \in J = \{1, ..., n\}$;
M : conjunto de m máquinas $i \in M = \{1, ..., m\}$;

A : $\{a_1, ..., a_i, ..., a_p\}$: conjunto das p restrições de precedência, onde a i-ésima restrição $a_i = (k,j)$ representa que a tarefa k deve preceder a tarefa j;

$A_{(k,j^*)}$: conjunto das restrições de precedência onde é considerado apenas as tarefas predecessoras a tarefa j^*;

H : conjunto contendo todas as tarefas já sequenciadas;

F : conjunto contendo todas as tarefas sequenciadas e finalizadas até o instante de tempo t;

D : conjunto de tarefas não sequenciadas e disponíveis para sequenciamento, onde todas as suas tarefas predecessoras pertencem ao conjunto de tarefas concluídas F;

T : conjunto dos tempos de conclusão em cada máquina $i \in M$;

t : instante de tempo do sequenciamento;

j^* : tarefa do conjunto D selecionada por uma regra de prioridade;

i^* : máquina selecionada por uma regra de prioridade;

l : última tarefa alocada em uma determinada máquina i;

z : maior tempo de conclusão das tarefas predecessoras de j^*;

L_i : tempo de conclusão da última tarefa alocada na máquina i;

C_j : tempo de conclusão da tarefa $j \in J$;

I_j : tempo de início de processamento da tarefa $j \in J$;

No Algoritmo 6 é descrito um pseudocódigo da heurística construtiva. A partir deste algoritmo obtém-se diferentes heurísticas construtivas variando apenas as regras de prioridade. No algoritmo, primeiramente, as variáveis u, H e L_i são inicializadas (linha 2). Em cada iteração u do algoritmo, uma tarefa j é sequenciada. O algoritmo termina quando todas as tarefas do conjunto J são sequenciadas, ou seja, quando $u = n$. Na linha 4, o conjunto T é formado pelos tempos de conclusão de cada máquina $i \in M$. Em seguida, na linha 5, o instante de tempo t do sequenciamento é determinado pelo menor tempo contido em T, ou seja, o menor valor de tempo de conclusão de todas as máquinas.

Nas Linhas 6 e 7, são calculados, respectivamente, o conjunto F contendo todas as tarefas finalizadas até o instante de tempo t, e o conjunto D, contendo todas as tarefas disponíveis para sequenciamento no instante t. Uma tarefa $j \in F$ é dita finalizada em um instante t, se o seu tempo de conclusão C_j é menor ou igual a t. Já uma tarefa $j \in D$ é dita disponível para sequenciamento em um instante t, se ela não estiver sequenciada e se todas as suas tarefas predecessoras estiverem concluídas, ou seja, todas as tarefas predecessoras devem estar no conjunto F.

Conforme as linhas de 8 a 13, se o conjunto de tarefas D estiver vazio, isto

é, quando não existem mais tarefas disponíveis no instante de tempo t, o valor de t passa a valer o próximo menor valor de tempo de conclusão contido em T e o conjunto D é recalculado. Esse processo se repete até que o conjunto D não fique vazio. Note que na primeira iteração ($u = 0$), o conjunto D sempre vai possuir algumas tarefas nas quais não possuem tarefas predecessoras, portanto, os passos das linhas de 8 a 13 não são executados na primeira iteração.

Algoritmo 6: HC

saída: solução construída S

```
1  início
2      u ← 0, H ← ∅, L_i ← 0, ∀i ∈ M;
3      enquanto u ≤ n faça
4          T := {L_i | i ∈ M};
5          t := min{L_i | L_i ∈ T};
6          F := {j | C_j ≤ t, ∀j ∈ H};
7          D := {j | ∀j ∈ (J − H), ∀k ∈ A_(k,j), k ∈ F};
8          enquanto D = ∅ faça
9              T := T \ {t};
10             t := min{L_i | L_i ∈ T};
11             F := {j | C_j ≤ t, ∀j ∈ H};
12             D := {j | ∀j ∈ (J − H), ∀k ∈ A_(k,j), k ∈ F};
13         fim
           /* selecionar uma tarefa j* ∈ D e uma máquina i* ∈ M   */
14         (j*, i*) := Regra_de_Prioridade(D, M);
           /* calcular tempo de conclusão e atualizar variáveis  */
15         z := max{C_k | ∀k ∈ A_(k,j*)};
16         C_j* := max{z, (R_i* + s_i*lj*)} + p_i*j*;
17         I_j* := C_j* − (s_i*lj* + p_i*j*);
18         L_i* := C_j*;
19         H := H ∪ {j*};
20         u := u + 1;
21     fim
22     C_max := max{C_j | j ∈ J};
23 fim
```

Na linha 14, será aplicada a regra de prioridade para selecionar uma tarefa $j^* \in D$ que será alocada na última posição de uma máquina $i^* \in M$, que também é selecionada pela regra de prioridade.

Na linha 15, o valor de z é calculado como sendo o tempo máximo de conclusão das tarefas predecessoras de j^*. Em seguida, nas linhas 16 e 17, são calculados o tempo de conclusão C_{j^*} e o tempo de início I_{j^*} da tarefa selecionada. Nas linhas 18 e 19, o tempo de conclusão da máquina i^* (R_{i^*}) passa a ser o tempo de conclusão

da tarefa j^* e a tarefa entra para o conjunto H, onde estão todas as tarefas já sequenciadas. Por fim, na linha 20, a variável u é incrementada para a próxima iteração. Quando todas as tarefas já estiverem sequenciadas, o algoritmo calcula valor do maior tempo de conclusão das tarefas $j \in J$, que é o *makespan* (C_{max}).

Como pode ser observado, o algoritmo pode ser resolvido em tempo polinomial. O código da linha 8 a 13 é executado quando não existem tarefas disponíveis para sequenciamento, sendo executado no máximo n iterações a cada iteração do algoritmo. Considerando que o algoritmo finaliza após n iterações quando todas as tarefas estão sequenciadas, então sua complexidade pode ser definida por $O(n^2)$.

Na Figura 5.1 é apresentado um diagrama de fluxo que resume todos os passos executados pelo algoritmo até formar uma solução completa.

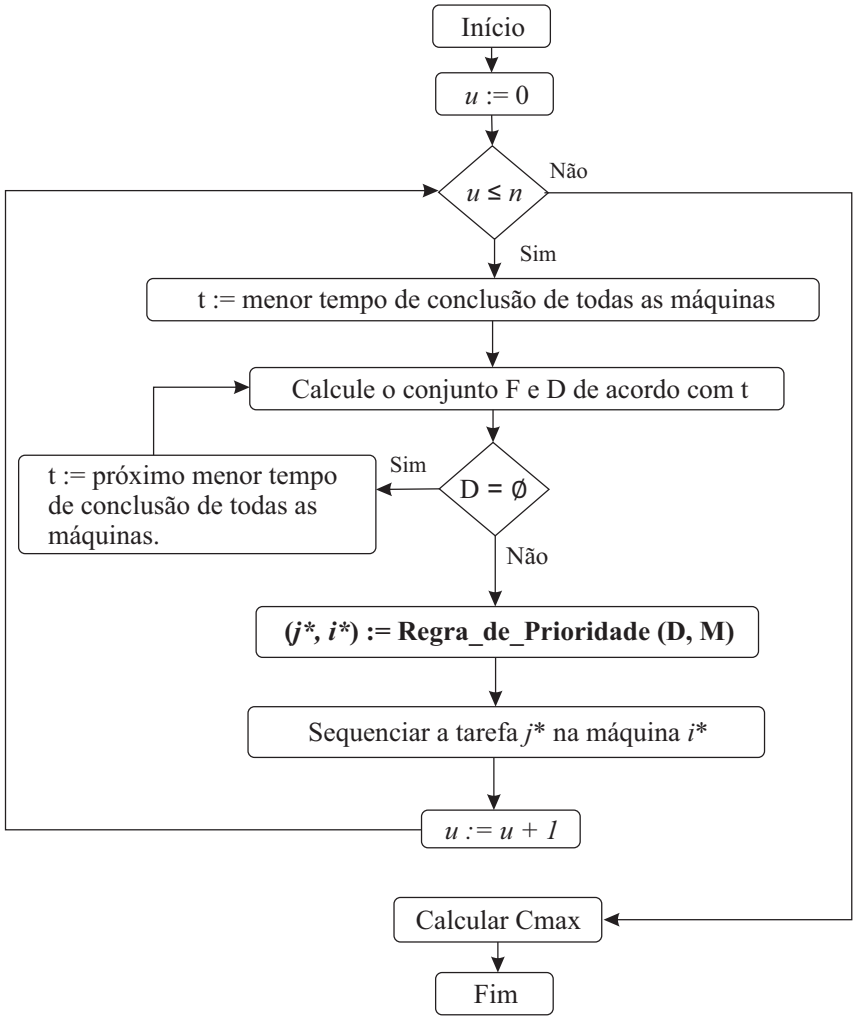


Figura 5.1. Diagrama de Fluxo da Heurística Construtiva.

Na próxima seção serão apresentadas sete regras de prioridade a serem utili-

zadas na heurística HC. As regras de prioridade são denotadas de $R1$ a $R7$.

5.1.1 Regras de Prioridade

Como visto na heurística HC (Algoritmo 6), em cada iteração aplica-se uma regra de prioridade para selecionar uma tarefa $j^* \in D$ e uma máquina $i^* \in M$ na qual a tarefa será alocada. Nesta seção serão apresentadas sete regras de prioridade denotadas de $R1$ a $R7$. Estas regras foram adaptadas de trabalhos da literatura sobre problemas de sequenciamento em máquinas paralelas. A adaptação consiste em considerar as restrições de precedência e os tempos de preparação tratados neste trabalho.

A seguir são descritas cada uma das regras de prioridade utilizadas:

- $R1$: adaptada de Weng et al. (2001), esta regra seleciona uma tarefa $j^* \in D$ que obtiver a menor média do tempo de processamento somado ao tempo de preparação, conforme equação 5.1. Depois será selecionada uma máquina $i^* \in M$ que gerar menor tempo de conclusão para sequenciar a tarefa j^* logo após a última tarefa da máquina. Dado que k é a última tarefa alocada em uma máquina i, L_i é o tempo de conclusão da tarefa k e z é o tempo de conclusão máximo entre as tarefas predecessoras de j^*, a regra se resume nas seguintes equações:

$$j^* = min\{p_j : j \in J\}, \quad p_j = \frac{\sum_{i=1}^{m} S_{ikj} + p_{ij}}{m}, \quad \forall j \in D \tag{5.1}$$

$$i^* = min\{max(z, (L_i + s_{ikj^*})) + p_{ij^*} : i \in M\} \tag{5.2}$$

- $R2$: adaptada de Weng et al. (2001), esta regra seleciona uma tarefa $j^* \in D$ da mesma forma que a Regra 1. No entanto, de acordo com a equação 5.3 e considerando k sendo a última tarefa alocada em uma máquina i, a máquina i^* selecionada será a máquina onde a tarefa j^* possuir menor tempo de processamento acrescido do tempo de preparação.

$$i^* = min\{S_{ikj^*} + p_{ij^*} : i \in M\} \tag{5.3}$$

- $R3$ e $R4$: em ambas as regras, a tarefa $j^* \in D$ será a tarefa que tiver menor tempo de processamento acrescido do tempo de preparação, definida pela equação 5.4. No entanto, na regra $R3$ a máquina é definida da mesma forma

que a regra $R1$ e, na regra $R4$, a máquina é selecionada da mesma forma que em $R2$.

$$j^* = min\{S_{ikj} + p_{ij} : j \in J, i \in M\} \tag{5.4}$$

- $R5$: utilizada por Lin et al. (2011), esta regra seleciona a tarefa $j^* \in D$ com menor tempo de conclusão se alocada no final de cada máquina $i \in M$. Depois, a tarefa j^* será alocada na máquina i^* que gerar o menor tempo de conclusão para j^*. Dado que L_i é o tempo de conclusão da última tarefa k alocada em i, os valores de j^* e i^* são definidos pelas seguintes equações:

$$j^* = min\{max(z, (L_i + s_{ikj})) + p_{ij} : j \in J, i \in M\} \tag{5.5}$$

$$i^* = min\{max(z, (L_i + s_{ikj})) + p_{ij^*} : i \in M\} \tag{5.6}$$

- $R6$: adaptada de Liu & Yang (2011), esta regra seleciona a tarefa $j^* \in D$ que obtiver o maior desvio padrão do tempo de processamento mais o tempo de preparação nas máquinas $i \in M$. A máquina i^* será a que gerar o menor tempo de conclusão para a tarefa j^*, conforme a equação 5.6. A seguir são apresentados o cálculo da média aritmética e do desvio padrão:

$$E_j = \frac{1}{|M|}\sum_{i=1}^{m}(p_{ij} + S_{ikj}), \quad \sigma_j^2 = \frac{1}{|M|}\sum_{i=1}^{m}((p_{ij} + S_{ikj}) - E_j)^2, \quad \forall j \in J \tag{5.7}$$

- $R7$: adaptada da regra $R5$, esta regra consiste em criar um conjunto L contendo uma certa porcentagem α de tarefas com menor tempo de conclusão nas máquinas de acordo com a equação 5.5. A tarefa selecionada $j^* \in L$, será a tarefa que preceder o maior número de tarefas de acordo com as restrições de precedência. A máquina i^* será a que gerar menor tempo de conclusão para j^*, conforme equação 5.6.

A regra $R7$, além de conceder prioridade às tarefas de menor tempo de conclusão, seleciona a tarefa que preceder o maior número de tarefas. Quando uma tarefa, de acordo com as precedências, tem maior número de tarefas sucessoras e é sequenciada primeiro, maior será a chance de aumentar o número de tarefas disponíveis em D nas próximas iterações do algoritmo. Com isso, tarefas com menor

tempo de conclusão poderão ser sequenciadas primeiro. A heurística HC e as regras de prioridade apresentadas geram sete heurísticas construtivas denotadas de $HC1$ a $HC7$.

5.2 Experimentos Computacionais sob as Heurísticas Construtivas

Nesta seção é apresentada a calibração do parâmetro α da heurística $HC7$, em seguida é analisado o desempenho das heurísticas construtivas $HC1$ a $HC7$ e, por fim, é comparada a heurística de melhor desempenho com o Modelo 1 apresentado no Capítulo 4. As heurísticas foram codificadas na linguagem C++ e os experimentos foram realizados em um computador com processador Intel Core i7 com 4.00 GHz de clock, 32GB de memória e sistema operacional Windows 8 64 bits.

5.2.1 Calibração do parâmetro α usado pela heurística HC7

O parâmetro α é utilizado pela heurística $HC7$ para definir o tamanho do conjunto L de tarefas candidatas à seleção. Este parâmetro é testado utilizando os seguintes valores: $\alpha(\%) = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. A heurística foi executada para todas as instâncias de grande porte. A métrica utilizada para avaliar os resultados dos experimentos é o Desvio Percentual Relativo (RPD) que é obtido pela seguinte equação:

$$RPD\% = \frac{f - f_{best}}{f_{best}} \times 100, \tag{5.8}$$

onde f corresponde ao valor da função objetivo obtido pela heurística e f_{best} corresponde ao melhor valor encontrado a partir da execução da heurística para todos os valores do parâmetro.

Para validar e comprovar a diferença estatisticamente significativa entre os resultados, foi utilizada a Análise de Variância (ANOVA) paramétrica (Miller Jr, 1997). Também foram verificadas as três pressuposições da ANOVA: normalidade, igualdade de variância e independência dos resíduos. Os testes apontam que há diferenças significativas para o uso dos diferentes valores de α e é possível observar que para $\alpha = 30\%$, obtém-se a menor média. Portanto, nos próximos experimentos, a heurística $HC7$ será executada usando $\alpha = 30\%$.

A Figura 5.2 mostra o gráfico de intervalos HSD de *Tukey* com nível de confiança de 95% para a comparação da heurística $HC7$ com os diferentes valor para

α. Como pode ser observado, para o valor de $\alpha = 0\%$, a heurística $HC7$ passa a selecionar a tarefa somente pelo menor tempo de conclusão. Para $\alpha = 100\%$, a heurística seleciona a tarefa somente pelo maior número de tarefas sucessoras. A melhora da média do RPD obtida entre os valor de $\alpha = 0\%$ a 30%, apesar de não ter apresentado diferença significativa, vem reforçar que a estratégia de priorizar também as tarefas com maior número de tarefas sucessoras pode melhorar a qualidade da solução.

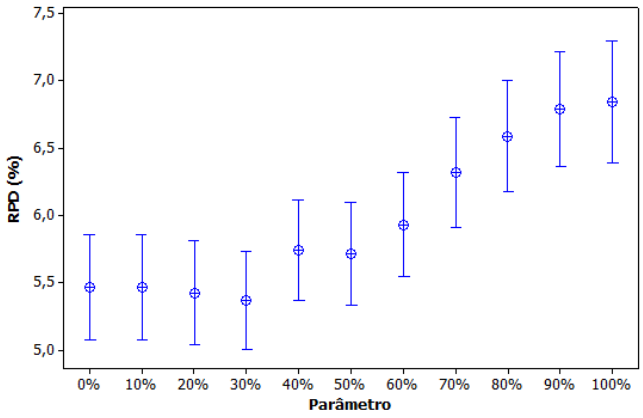


Figura 5.2. Gráfico de Intervalos HSD de *Tukey* para Calibração do Parâmetro α.

5.2.2 Comparação entre as heurísticas propostas

Neste experimento é analisado o desempenho das heurísticas propostas de $HC1$ a $HC7$. O experimento utilizou o conjunto de instâncias de grande porte e o RPD é calculado considerando a melhor solução obtida dentre todas as sete heurísticas. Também foi realizada a Análise de Variância Paramétrica (ANOVA), que apontou diferença significativa entre as sete heurísticas.

Na Figura 5.3 pode ser observado o gráfico de intervalos HSD de *Tukey* com nível de confiança de 95%. Como pode ser observado, a heurística $HC5$, com média de RPD = 3,87%, e a heurística $HC7$, com média de RPD = 3,71%, foram as heurísticas com melhor desempenho.

5.2.3 Comparação entre a heurística HC7 e o Modelo 1

Neste experimento é comparado o desempenho entre o Modelo 1, a heurística $HC7$ e um método onde as soluções são geradas aleatoriamente. No método aleatório,

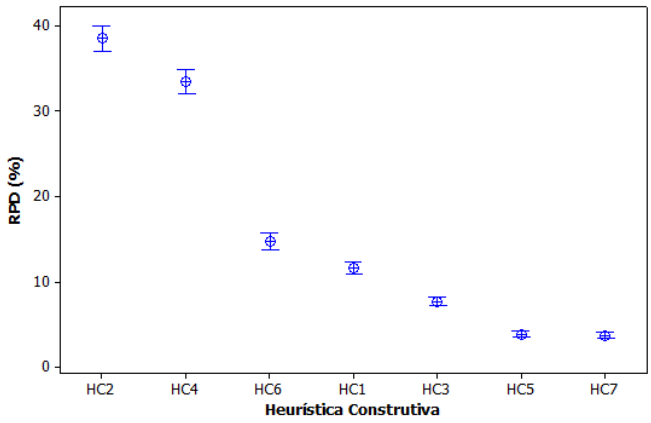


Figura 5.3. Gráfico de Intervalos HSD de *Tukey* para Comparação entre as Heurísticas.

para cada instância, geram-se 50 soluções aleatórias e escolhe-se a melhor. Para as comparações é utilizado o conjunto de instâncias de tamanho médio e o modelo matemático é resolvido pelo CPLEX com limite máximo de 3600 segundos de tempo de CPU.

A Tabela 5.1 é dividida em 3 partes. Na primeira parte, a média do valor de C_{max} é comparado entre o modelo, a heurística e o método aleatório para os 12 grupos de instâncias. Na média de cada grupo, é possível observar que o Modelo 1 obteve soluções com menor valor para C_{max}. A heurística $HC7$ obteve soluções mais próximas do Modelo 1 do que o método aleatório e também conseguiu encontrar soluções melhores que o Modelo 1 para 35 instâncias. Ou seja, o Modelo 1, no tempo de 3600 segundos (1 hora), não consegue determinar a solução ótima para algumas instâncias.

Tabela 5.1. Comparação entre o Modelo 1, $HC7$ e Soluções Aleatórias.

			Média C_{max}			Modelo 1		Média RPD (%)	
Grupo	n	m	Modelo 1	$HC7$	Aleatório	Gap_{LB}	Tempo (s)	$HC7$	Aleatório
1.	15	2	**409,31**	517,36	632,69	5,24	853,46	**26,40**	55,70
2.	15	5	**169,89**	207,78	401,36	**0,00**	2,45	**23,15**	146,15
3.	15	10	**99,02**	117,09	314,78	**0,00**	1,08	**18,62**	238,84
4.	20	2	**545,71**	653,36	888,53	26,69	2.744,57	**20,06**	65,83
5.	20	5	**201,31**	253,02	551,07	0,29	214,72	**25,29**	186,43
6.	20	10	**126,09**	154,96	436,62	**0,00**	6,63	**23,90**	269,48
7.	25	2	**691,36**	820,31	1.161,20	36,39	3.122,13	**18,53**	70,76
8.	25	5	**252,60**	309,16	728,71	10,68	1.689,19	**24,39**	209,23
9.	25	10	**135,47**	162,84	536,47	**0,00**	95,08	**23,41**	325,48
10.	30	2	**814,27**	924,82	1.362,36	50,01	3.528,48	**14,38**	69,12
11.	30	5	**313,98**	351,86	886,27	29,85	3.122,53	**15,38**	195,80
12.	30	10	**189,16**	203,80	689,69	0,79	738,27	**13,89**	291,21
	Médias		**329,01**	389,70	715,81	13,33	1.343,27	**20,62**	177,00

Na segunda parte da tabela é apresentada a média do Gap_{LB}. O valor do Gap_{LB} avalia a diferença entre o valor do C_{max} e o valor do *lower bound* obtido pelo Modelo 1 para uma determinada instância. O Gap_{LB} do modelo foi em média 18,60% e os grupos 2, 3 e 6 obtiveram média de 0%, o que significa que todas as soluções encontradas para as instâncias desses grupos são ótimas. Com limite máximo de 3.600 segundos de execução para cada instância, o Modelo 1 gastou em média 1.821,98 segundos e conseguiu encontrar 299 soluções ótimas do total de 540 instâncias. Já a heurística $HC7$ gastou 0,001 segundos para construir uma solução para cada instância, conseguindo encontrar apenas 9 soluções ótimas das 299 encontradas pelo modelo.

Na última parte da tabela, o RPD médio da heurística e do método aleatório são comparados. O RPD é calculado considerando a melhor solução encontrada da heurística $HC7$ e do Modelo 1. Pode ser observado que $HC7$ obteve RPD médio total de 20,62% contra 177,00% das soluções aleatórias.

Na Tabela 5.2 é apresentado uma comparação entre o Modelo 1 utilizando a heurística $HC7$ como solução inicial e o Modelo 1 sem solução inicial. Ambos foram executados utilizando 600 segundos de tempo limite. Como pode ser observado, o Modelo 1 com a heurística $HC7$ obteve os melhores valores para a média do C_{max}, do Gap (%) e do tempo de execução. Além disso, o Modelo 1 com a heurística $HC7$ obteve 337 soluções ótimas contra 324 do Modelo 1 sem solução inicial. Nos próximos capítulos, os resultados do Modelo 1 utilizando a heurística $HC7$ como solução inicial será comparado com outros métodos de resolução implementados.

Tabela 5.2. Comparação entre o Modelo 1 sem solução inicial e o Modelo 1 com a solução da heurística $HC7$ como solução inicial.

		Modelo 1			HC7 + Modelo 1		
n	m	C_{max}	Gap (%)	Tempo (s)	C_{max}	Gap (%)	Tempo (s)
15	2	**409,93**	**9,70**	**250,22**	**409,93**	10,24	252,70
15	5	**169,89**	**0,30**	2,48	**169,89**	0,39	**1,35**
15	10	**99,02**	0,58	1,09	**99,02**	**0,55**	**0,19**
20	2	551,78	30,32	480,32	547,58	**30,23**	**478,31**
20	5	**196,13**	**1,46**	107,33	196,58	1,61	**96,54**
20	10	**126,09**	**0,13**	5,36	126,09	0,59	**1,00**
25	2	720,11	39,96	**532,08**	**715,20**	**38,92**	534,99
25	5	255,07	15,81	396,39	**242,93**	**12,08**	**358,40**
25	10	**130,93**	0,30	85,16	**130,93**	**0,05**	**10,98**
30	2	904,07	52,56	592,64	**859,00**	**50,61**	**591,05**
30	5	391,27	41,75	560,35	**310,04**	**30,59**	**552,86**
30	10	205,47	17,90	303,04	**161,58**	**2,48**	**149,65**
Médias		346,65	17,56	276,37	**330,81**	**14,87**	**252,33**

5.3 Conclusão

Neste capítulo foi apresentada uma heurística construtiva baseada em regra de prioridade, onde foram propostas sete diferentes regras de prioridade. De acordo com os experimentos computacionais realizados, as heurísticas de $HC1$ a $HC7$ apresentaram diferenças significativas em relação a qualidade das soluções geradas, evidenciando que o desempenho da heurística construtiva depende da regra de prioridade aplicada. Conforme o experimento, a heurística $HC7$ obteve menor média de RPD em relação as demais heurísticas. Em seguida, a heurística $HC7$ foi comparada com o Modelo 1 e com soluções geradas aleatoriamente. A heurística obteve resultados significativamente melhores que as soluções geradas aleatoriamente e, em relação ao Modelo 1, a heurística conseguiu ser melhor apenas para poucas de instâncias, mas obteve um RPD relativamente baixo se comparado com as soluções aleatórias.

Capítulo 6

Proximity Search Aplicado ao Problema $R_m \mid prec, S_{ijk} \mid C_{max}$

Como visto no Capítulo 3, o *Proximity Search* é uma técnica usada para resolver problemas de programação linear inteira mista (PLIM) proposta por Fischetti & Monaci (2014). A sua principal característica é obter iterativamente uma sequência de soluções melhoradas aplicando uma função de proximidade e uma restrição de corte sobre um modelo de PLIM.

Neste capítulo são apresentadas três diferentes implementações da heurística *Proximity Search* para resolver o problema proposto neste trabalho. O funcionamento de cada implementação e os experimentos computacionais realizados serão discutidos nas próximas seções.

6.1 Função de Proximidade

O *Proximity Search* (PS) tem como objetivo encontrar uma solução X melhor que a solução corrente X', cuja diferença entre essas duas soluções seja a menor possível. A diferença entre as duas soluções é calculada a partir da métrica conhecida como distância de *Hamming*. No PS, essa métrica é usada para compor a função de proximidade.

A função de proximidade, denotada por $\Delta(X, X')$, vai substituir a função objetivo do modelo matemático utilizado internamente pelo PS. O objetivo do modelo não é mais encontrar a solução de menor *makespan*, mas é encontrar uma solução X melhor e mais semelhante à solução corrente X'. Essa característica torna a resolução de um modelo matemático mais fácil de ser resolvido logo nas

primeiras iterações do PS, obtendo assim uma quantidade maior de melhorias em pouco tempo computacional.

Considere M o conjunto de m máquinas, J o conjunto de n tarefas e $J_0 = J \cup \{0\}$ onde 0 é uma tarefa fictícia que é usada para ser a primeira tarefa da sequência de cada máquina. A função de proximidade mede a "distância" entre as soluções X e X' definidas, respectivamente, por:

$$X = \{x_{ijk} : \forall i \in M, j \in J_0, k \in J\} \tag{6.1}$$

$$X' = \{x'_{ijk} : \forall i \in M, j \in J_0, k \in J\} \tag{6.2}$$

onde x_{ijk} e x'_{ijk} são as variáveis do modelo. Estas variáveis possuem valor igual a 1 se a tarefa k é processada imediatamente após a tarefa j na máquina i, e 0, caso contrário.

A função de proximidade para o problema proposto pode ser definida pela seguinte equação:

$$\Delta(X, X') = \sum_{i \in M} \sum_{j \in J_0} \sum_{k \in J : x'_{ijk} = 0} x_{ijk} + \sum_{i \in M} \sum_{j \in J_0} \sum_{k \in J : x'_{ijk} = 1} (1 - x_{ijk}) \tag{6.3}$$

Aplicando o cálculo da função de proximidade, o valor de $\Delta(X, X')$ consiste em identificar a quantidade de valores distintos entre x_{ijk} e x'_{ijk}. Por exemplo, se na solução X a tarefa 4 está alocada logo após a tarefa 5 na máquina 1 ($x_{154} = 1$) e na solução X' isso não é verdadeiro ($x'_{154} = 0$), então o valor de $\Delta(X, X')$ será incrementado em 1. Também, se as soluções X e X' são iguais (os valores de todas suas variáveis são iguais), então $\Delta(X, X') = 0$.

6.2 Implementações

A seguir são apresentadas três diferentes implementações do *Proximity Search*, denotadas por $PS1$ e $PS2$.

6.2.1 $PS1$

Nesta implementação, o $PS1$ utiliza o Modelo 1 como base para construir o modelo matemático a ser resolvido em cada iteração do algoritmo. O modelo é resolvido utilizando o software IBM ILOG CPLEX 12.6.2.

Em cada iteração do $PS1$, a função objetivo do Modelo 1 é substituída pela função de proximidade $\Delta(X, X')$, conforme descrita anteriormente, e uma restrição de corte é adicionada ao conjunto de restrições. Essa restrição é definida pela seguinte equação:

$$C_{max}(X) \leq C_{max}(X') - \theta \tag{6.4}$$

A restrição de corte vai reduzir o espaço de busca do modelo e o valor de θ vai definir o quanto melhor deve ser a solução X em relação a X'. Por exemplo, dado que $C_{max}(X') = 100$ e $\theta = 5$, então o valor de $C_{max}(X)$ da solução a ser encontrada deve ter *makespan* menor ou igual a 95.

No Algoritmo 7 é exemplificado o funcionamento do $PS1$. Na linha 2, o valor de θ é definido igual a 1, que permanece constante durante todas as iterações algoritmo. Em seguida, na linha 3, a solução inicial é definida pela heurística construtiva $HC7$.

Nas linhas 5 e 6, respectivamente, a restrição de corte é adicionada ao modelo e a função objetivo é substituída pela função de proximidade calculada conforme visto na seção 6.1.

Na linha 7, o modelo é resolvido através do CPLEX, que é executado usando sua configuração padrão. A execução do modelo é abortada assim que a primeira solução viável é obtida ou quando o tempo limite restante definido para o $PS1$ é atingido. A solução X^* obtida pelo modelo será uma pequena melhora em relação a X' devido a restrição de corte e a função de proximidade.

Nas linhas de 8 a 13 é verificado se o modelo encontrou uma solução X^*. Se o modelo não encontrar uma solução, o algoritmo é finalizado e a melhor solução obtida é retornada. Caso contrário, a função objetivo $C_{max}(X^*)$ do modelo original é calculada e a função de proximidade é atualizada fazendo $X' = X^*$. De forma iterativa, o algoritmo retorna a linha 4 repetindo todas as etapas descritas. Caso o tempo limite seja atingido, o algoritmo é finalizado e a melhor solução obtida é retornada.

O $PS1$ consegue provar a otimalidade de uma solução somente se a resolução do modelo não encontrar nenhuma solução antes de atingir o tempo limite do algoritmo e o valor de θ na restrição de corte seja igual a 1. Nesta situação, não existe nenhuma solução melhor que X' para o corte aplicado, então X' é uma solução ótima. Vale lembrar que, neste algoritmo, a otimalidade só é possível ser provada com o valor de θ igual a 1. Por exemplo, dado que $C_{max}(X') = 100$ e $\theta = 5$ e que não existe uma solução X para $C_{max}(X) \leq 95$. Nesta situação não pode ser

Algoritmo 7: $PS1$

saída: uma solução viável X';
início
  $\theta := 1$;
  $X' :=$ solução inicial viável gerada pela heurística $HC7$;
  enquanto *tempo limite não tenha sido atingido* **faça**
    Adicionar ao Modelo 1 a restrição de corte:
    $C_{max}(X) \leq C_{max}(X') - \theta$;
    Substituir a função objetivo $C_{max}(X)$ do Modelo 1 pela função de proximidade $\Delta(X, X')$;
    $X^* :=$ a primeira solução viável encontrada ao resolver o Modelo 1;
    se $X^* \,!= \emptyset$ **então**
      $X^* := argmin\{C_{max}(X) : \textit{restrições (4.2) a (4.10)}, X = X^*\}$
      Atualizar $\Delta(X, X')$ fazendo $X' = X^*$;
    senão
      Pare;
    fim
  fim
fim

afirmado que X' é uma solução ótima, pois pode existir uma solução X com valor de *makespan* entre $95 < C_{max}(X) < 100$. Para $\theta = 1$, o corte seria de $C_{max}(X) \leq 99$ e considerando que S_{ijk} e p_{ij} possuem valores inteiros, neste caso, se não existir uma solução X, pode-se afirmar que X' é uma solução ótima. Se o algoritmo atingir o tempo limite, a melhor solução obtida pelo $PS1$ será apenas uma solução viável.

6.2.2 $PS2$ e $PS2_{RINS}$

O $PS2$ é uma variação do $PS1$ onde a restrição de corte é aplicada de forma mais suave, fazendo com que a solução X a ser encontrada possa ser igual à solução X'. Isso faz com que a solução incumbente do modelo seja inicializada por X' logo no início da resolução do modelo. No entanto, essa situação não é desejada, pois é necessário encontrar uma solução X diferente de X' e que sejam semelhantes. Para isso, a função de proximidade da solução a ser encontrada pelo modelo é penalizada. A principal vantagem de se inicializar a solução incumbente do modelo é ter uma solução incumbente logo no início para que sejam feitas podas na árvore de busca e para que sejam aplicadas heurísticas internamente implementadas pelo CPLEX.

A implementação do $PS2$ possui a mesma estrutura do $PS1$, no entanto, a

restrição de corte (6.4) é substituída por:

$$C_{max}(X) \leq C_{max}(X') - \theta + z \tag{6.5}$$

onde z é uma variável de folga contínua ($z \geq 0$). Nesta implementação, o valor de θ também é igual a 1 em todas as iterações do algoritmo. A função de proximidade também é alterada conforme a seguinte equação:

$$\Delta(X, X') + Mz \tag{6.6}$$

onde M é um valor positivo relativamente grande se comparado com o valor obtido por Δ.

Com a variável de folga z aplicada na função de proximidade e na restrição de corte, o valor da solução incumbente do modelo é rapidamente definida e pode ser igual à solução corrente X'. Essa situação seria inválida, pois espera-se encontrar uma solução X melhor e diferente de X'. Para isso, a constante M vai penalizar a solução com uma função de proximidade muito alta. Ter uma solução incumbente logo no início da resolução do modelo pode melhorar o desempenho e permitir a utilização de heurísticas de refinamento implementadas internamente pelo CPLEX.

Portanto, também é criada a versão denominada $PS2_{RINS}$ que utilizada a heurística de refinamento RINS (*Relaxation Induced Neighborhood Search*), introduzida por Danna et al. (2005). A heurística RINS aplica a ideia de fixar algumas das variáveis inteiras de uma determinada solução X. Em determinados nós da árvore do *branch-and-bound*, a solução do relaxamento contínuo de um nó (X_{rel}) e a solução incumbente (X_{inc}) são comparados. Para todas as variáveis com restrições de inteiros cujo valores são iguais tanto para X_{inc} quanto para X_{rel} são fixados e, para o restante das variáveis, o modelo é novamente resolvido. O CPLEX possui implementada a heurística RINS. Portanto, nesta implementação a heurística é habilitada e configurada para ser executada a cada nó encontrado, conforme o seguinte comando: cplex.setParam(IloCplex::RINSHeur, 1) da biblioteca ILOG Concert do CPLEX.

A execução do modelo é interrompida quando uma solução com $z = 0$ for encontrada. Isso significa que foi encontrada uma solução melhor que a solução atual. O $PS2$ e o $PS2_{RINS}$ vão executar até o que o tempo limite não tenha sido atingido ou até que não seja encontrada nenhuma solução X ao fim da execução do modelo matemático.

6.3 Experimentos Computacionais

Nesta seção são apresentados os experimentos computacionais realizados a fim de demostrar o desempenho das implementações do *Proximity Search* em relação ao Modelo 1. Nos experimentos, as versões do PS e o Modelo 1 foram implementados usando a biblioteca ILOG Concert, que possibilitou o desenvolvimento através da linguagem C++ e a resolução do modelo através do software IBM ILOG CPLEX 12.6.2.

A configuração do computador utilizado para os experimentos é a mesma citada no Capítulo 4. Em todos os experimentos, o $PS1$, $PS2$, $PS2_{RINS}$ e o Modelo 1 foram limitados a 600 segundos de tempo de CPU para cada instância. Além disso, foram acrescentadas instâncias com $n = 35$ e $n = 40$ tarefas no grupo de instâncias de tamanho médio a fim de demonstrar o bom desempenho do PS para instâncias maiores. Portanto, no total foram 810 instâncias com $n \in \{15, 20, 25, 30, 35, 40\}$ e $m \in \{2, 5, 10\}$. Tanto o Modelo 1 quanto o PS são inicializados com uma solução inicial gerada pela heurística construtiva $HC7$ apresentada no Capítulo 5.

Para avaliar os resultados obtidos foi calculado o RPI (*Relative Percentage Improve*) das soluções obtidas com relação à solução inicial obtida pela heurística $HC7$. Ou seja, essa métrica é usada para avaliar a porcentagem de melhoria provocada pelo método em relação a solução inicial fornecida. A métrica RPI pode ser definida como:

$$RPI(\%) = \frac{f_{s.i.} - f_{método}}{f_{s.i.}} \times 100, \tag{6.7}$$

onde $f_{s.i.}$ é o valor da função objetivo da solução inicial obtida pela heurística $HC7$, e $f_{método}$ é o valor da função objetivo da solução obtida pelo $PS1$, $PS2$, $PS2_{RINS}$ ou Modelo 1.

Outra métrica utilizada para avaliar o PS é chamada de *primal integral* que será descrita com mais detalhes a seguir.

6.3.1 Métrica de Comparação *Primal Integral*

A métrica denominada *primal integral*, proposta por Achterberg et al. (2012) e Berthold (2013), é utilizada para comparar o desempenho entre o PS e a resolução do Modelo 1. Essa métrica tem como objetivo avaliar o *trade-off* entre o esforço computacional gasto pelo método para encontrar uma solução e a qualidade da própria solução.

Considerando que z_{opt} é a solução ótima ou a melhor solução conhecida para uma determinada instância do problema e $z(t)$ o valor da melhor solução encontrada até o instante de tempo t, o *primal gap function p(t)*, cujo valor deve estar no intervalo [0,1], pode ser calculado da seguinte forma:

$$p(t) = \begin{cases} 1, & \text{Se nenhuma solução incumbente foi encontrada até o instante de tempo } t \\ \frac{z(t)-z_{opt}}{z_{opt}}, & \text{Caso contrário.} \end{cases} \quad (6.8)$$

Em seguida, o *primal integral* é executado de $t = 0$ até t_{max}, onde t_{max} é o tempo limite de execução do método. O *primal integral* é calculado conforme a seguinte equação:

$$P(t_{max}) = \int_0^{t_{max}} p(t) \times \Delta t \quad (6.9)$$

A integral calculada por esta equação vai definir a área formada por $p(t) \times \Delta t$. O valor de Δt representa o intervalo de tempo gasto entre a última melhoria e a melhoria atual da solução corrente.

O *primal integral* pode ser usado para medir a qualidade de diferentes heurísticas em relação a rapidez com que as melhorias são obtidas sobre uma determinada solução. Para definir se uma heurística é melhor que a outra usando essa métrica, deve-se observar o valor de $P(t)$ partindo de t_{max} até $t = 0$. Quanto menor é o valor assumido por $P(t)$, melhor é o desempenho da heurística.

6.3.2 Análise dos Resultados Obtidos pelo PS

Como já mencionado, o Modelo 1 e todas as versões do PS iniciam a partir de uma solução inicial gerada pela heurística construtiva $HC7$. Para avaliar o quanto os métodos implementados melhoraram a solução inicial, foi calculado o RPI de cada instância do problema.

Na primeira parte da Tabela 6.1 é apresentada a média de RPI(%) de cada configuração $n \times m$ das instâncias obtida pelo Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$. Como pode ser observado, o $PS2_{RINS}$ obteve a maior média total de RPI (20,16%), ou seja, o $PS2_{RINS}$ obteve maior porcentagem de melhoramento da solução inicial. Já o $PS1$, $PS2$ e o Modelo 1 obtiveram, respectivamente, RPI igual a 18,32%, 18,99% e 13,45%.

Na segunda parte da Tabela 6.1 é apresentado o tempo médio em segundos

gasto pelo Modelo 1 e todas as versões do PS. Como pode ser observado, o $PS2$ foi o que apresentou o menor tempo médio.

Tabela 6.1. RPI(%) e tempo médio (s) gasto pelo Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$.

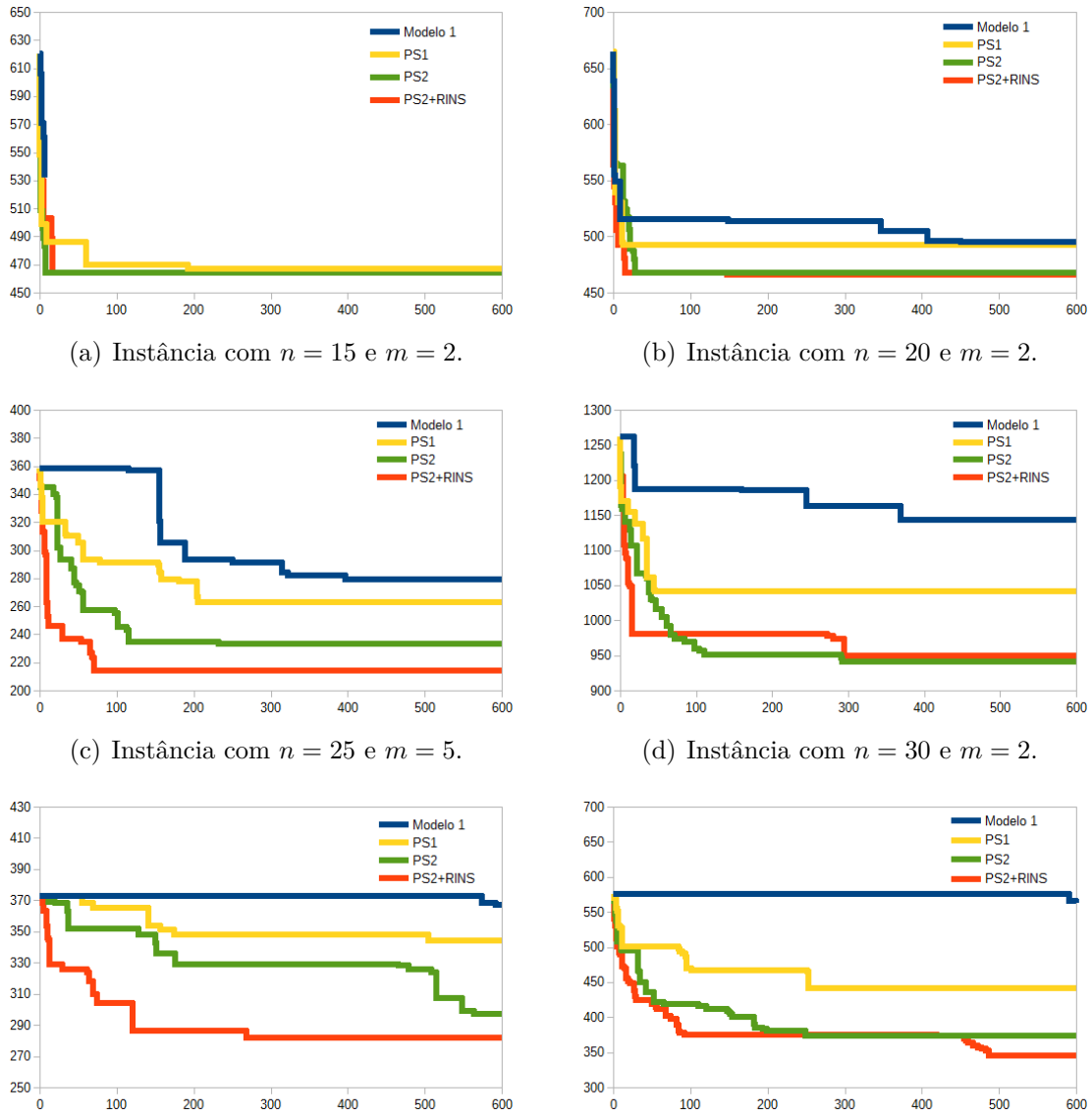
n	m	RPI (%)				Tempo (s)			
		Modelo 1	$\boldsymbol{PS1}$	$\boldsymbol{PS2}$	$\boldsymbol{PS2_{RINS}}$	**CPLEX**	$\boldsymbol{PS1}$	$\boldsymbol{PS2}$	$\boldsymbol{PS2_{RINS}}$
15	2	19,60	**20,11**	20,03	19,92	252,70	261,16	**149,98**	499,82
15	5	**17,38**	**17,38**	17,15	**17,38**	**1,35**	1,85	1,86	29,25
15	10	**14,61**	**14,61**	14,59	**14,61**	**0,19**	1,30	1,09	0,99
20	2	15,89	17,07	17,44	**17,60**	**478,31**	486,68	481,16	583,10
20	5	20,98	**21,16**	21,13	21,10	96,54	81,16	**40,59**	331,49
20	10	**18,05**	**18,05**	**18,05**	**18,05**	**1,00**	4,05	3,89	16,42
25	2	12,12	17,47	17,37	**18,68**	534,99	562,43	**534,86**	591,47
25	5	20,98	22,60	22,67	**23,73**	358,40	321,90	**291,01**	509,72
25	10	**19,66**	**19,66**	19,63	**19,66**	**10,98**	12,91	11,90	134,22
30	2	7,07	15,67	16,81	**18,22**	**591,05**	600,00	600,00	600,00
30	5	12,02	21,16	22,31	**24,58**	552,86	537,55	**526,95**	580,44
30	10	20,59	**21,34**	21,28	21,31	149,65	87,98	**79,59**	357,97
35	2	3,98	14,84	16,72	**19,05**	**600,00**	**600,00**	**600,00**	**600,00**
35	5	6,90	17,64	19,53	**23,25**	566,59	563,72	**548,27**	586,81
35	10	15,85	20,06	20,51	**20,69**	378,18	287,68	**274,34**	494,26
40	2	1,77	12,53	13,76	**17,30**	**600,00**	**600,00**	**600,00**	**600,00**
40	5	4,13	19,07	21,91	**25,07**	590,37	579,42	**576,60**	600,00
40	10	10,56	19,42	20,97	**22,60**	486,32	441,50	**433,78**	534,00
Médias:		13,45	18,32	18,99	**20,16**	347,32	335,12	**319,81**	425,00

Analisando o número de soluções ótimas obtidas por cada método implementado, do total de 810 instâncias, o $PS2$ foi o que encontrou o maior número de soluções ótimas, totalizando 418 soluções. Já o Modelo 1, o $PS1$ e o $PS2_{RINS}$ obtiveram, respectivamente, 373, 386 e 265 soluções ótimas.

A partir desses resultados, pode-se concluir que o $PS2_{RINS}$ conseguiu obter as melhores soluções dentro do tempo limite de 600 segundos. No entanto, possui maior tempo médio gasto para finalizar a execução dos problemas e menor número de soluções ótimas encontradas. Já o $PS2$ obteve menor tempo médio gasto para resolver as instâncias e encontrou o maior número de soluções ótimas. No entanto, obteve RPI médio um pouco inferior ao $PS2_{RINS}$. Vale lembrar que o $PS2_{RINS}$ se difere do $PS2$ apenas na utilização da heurística $RINS$. Essa heurística é configurada para ser aplicada a cada nó da árvore resultante da resolução do modelo matemático. Essa configuração pode ser alterada informando o valor do intervalo de nós em que a heurística deve ser aplicada. Uma calibração desse valor pode ajudar a melhorar o desempenho em relação ao tempo gasto pelo $PS2_{RINS}$.

A principal característica do PS é encontrar uma quantidade maior de soluções melhoradas logo nos primeiros instantes de tempo do algoritmo. Para reforçar essa teoria, foram selecionadas 6 instâncias com diferentes quantidades de tarefas. A

cada melhora obtida pelo PS foi armazenado o tempo gasto para obter a solução. Ao final, foi gerado o gráfico $tempo(s) \times C_{max}(X)$ para cada instância. O resultado é apresentado na Figura 6.1.



(a) Instância com $n = 15$ e $m = 2$.

(b) Instância com $n = 20$ e $m = 2$.

(c) Instância com $n = 25$ e $m = 5$.

(d) Instância com $n = 30$ e $m = 2$.

(e) Instância com $n = 35$ e $m = 5$.

(f) Instância com $n = 40$ e $m = 5$.

Figura 6.1. Atualização da solução de diferentes instâncias ao decorrer de 600 segundos ($Tempo(s) \times C_{max}(X)$) usando o Modelo 1, $PS1$ e $PS2$.

Como pode ser observado na Figura 6.1, as versões do PS conseguem encontrar soluções melhores com o menor tempo de execução se comparadas ao Modelo 1. E, ao final de 600 segundos, conseguem obter soluções com menor valor de $C_{max}(X)$.

Para analisar o desempenho das versões do PS e do Modelo 1 em relação o tempo gasto para obter cada solução melhorada, é calculada a métrica *Primal In-*

tegral para todas as instância do problema em um determinado intervalo de tempo de execução. Esses intervalos são definidos em 5, 10, 30, 60, 120, 300 e 600 segundos. Quanto menor é o valor da integral em cada intervalo de tempo, melhor é o desempenho do método implementado. A Tabela 6.2 apresenta a média geométrica da Integral $P(t)$ de todas as instâncias para cada método aplicado e em cada intervalo de tempo. Em todos os resultados apresentados, o $PS2_{RINS}$ foi o método que apresentou a menor média geométrica da integral em todos os intervalos de tempo.

Tabela 6.2. Média geométrica do *primal integral* de todas as instâncias depois de 5, 10, 30, ..., 600 segundos (Quanto Menor Melhor).

Tempo (s)	**Modelo 1**	$\boldsymbol{PS1}$	$\boldsymbol{PS2}$	$\boldsymbol{PS2_{RINS}}$
5	0,537258	0,602507	0,580623	**0,507155**
10	0,829489	0,878808	0,848881	**0,699505**
30	1,567762	1,470524	1,388857	**1,032072**
60	2,287477	1,945543	1,783621	**1,234996**
120	3,261667	2,477806	2,214210	**1,433289**
300	5,042065	3,283647	2,843790	**1,670650**
600	6,791820	3,943859	3,359076	**1,809259**

Na Tabela 6.3 é apresentada a média geométrica da integral ($P(t)$) de todas as instâncias para cada grupo de instâncias ($n \times m$), para cada método e para cada intervalo de tempo. Também pode ser observado que o $PS2_{RINS}$ obteve os menores valores das médias geométricas.

Na Figura 6.2 é apresentado o gráfico da média geométrica da integral de cada método implementado. Como pode ser observado, o Modelo 1 apresentou uma curva mais aberta, ou seja, possui os maiores valores para a média geométrica da integral. Já as versões do PS apresentaram uma curva mais acentuada, em especial o $PS2_{RINS}$, que apresentou uma diferença significativa.

6.4 Conclusão

Neste capítulo foram implementadas três diferentes versões do PS, denotadas de: $PS1$, $PS2$ e $PS2_{RINS}$. O $PS1$ é uma versão básica do PS onde o Modelo 1 é resolvido de forma iterativa, sempre substituindo a função objetivo pela função proximidade e também adicionando uma restrição de corte onde o parâmetro θ é fixada igual a 1. O $PS2$ é semelhante ao $PS1$, porém, permite que o modelo matemático inicie com uma solução incumbente através do uso de uma variável de folga z e de uma variável M, que é utilizada para penalizar uma possível solução inviável. No $PS2_{RINS}$, o $PS2$ é acrescido de uma heurística de melhoramento internamente implementada pelo CPLEX, conhecida como RINS.

Tabela 6.3. Média Geométrica do Primal Integral depois de 5, 10, 30, ..., 600 segundos (Quanto Menor Melhor).

			Tempo (s)						
n	m		**5**	**10**	**30**	**60**	**120**	**300**	**600**
25	2	$Modelo\,1$	0,917	1,706	4,206	7,186	11,960	22,553	35,618
		$PS1$	0,649	1,026	2,040	3,030	4,236	6,324	8,402
		$PS2$	0,698	1,105	2,061	2,954	4,075	6,207	8,316
		$PS2_{RINS}$	**0,557**	**0,833**	**1,506**	**2,098**	**2,792**	**3,854**	**4,542**
25	5	$Modelo\,1$	0,820	1,349	2,876	4,344	6,148	8,948	11,031
		$PS1$	0,823	1,176	1,932	2,551	3,128	3,874	4,378
		$PS2$	0,769	1,138	1,805	2,275	2,829	3,591	4,290
		$PS2_{RINS}$	**0,608**	**0,771**	**1,030**	**1,204**	**1,389**	**1,644**	**1,785**
25	10	$Modelo\,1$	**0,274**	**0,316**	**0,341**	**0,343**	**0,343**	**0,343**	**0,343**
		$PS1$	0,569	0,705	0,763	0,764	0,764	0,764	0,764
		$PS2$	0,501	0,607	0,657	0,662	0,668	0,679	0,688
		$PS2_{RINS}$	0,466	0,549	0,573	0,573	0,573	0,573	0,573
30	2	$Modelo\,1$	1,058	2,065	5,868	11,054	20,887	48,217	87,574
		$PS1$	0,830	1,475	3,524	5,765	9,069	16,243	24,119
		$PS2$	0,817	1,431	3,060	4,602	6,754	10,701	14,859
		$PS2_{RINS}$	**0,696**	**1,064**	**1,895**	**2,662**	**3,683**	**5,565**	**7,110**
30	5	$Modelo\,1$	1,340	2,519	6,752	12,285	21,188	42,500	68,432
		$PS1$	1,237	2,076	4,235	6,186	8,699	13,157	17,316
		$PS2$	1,111	1,855	3,798	5,455	7,519	11,424	14,783
		$PS2_{RINS}$	**0,985**	**1,471**	**2,412**	**3,055**	**3,732**	**4,595**	**5,023**
30	10	$Modelo\,1$	**0,657**	**0,949**	1,515	1,902	2,256	2,538	2,658
		$PS1$	0,855	1,285	1,868	2,034	2,077	2,078	2,078
		$PS2$	0,819	1,192	1,703	1,878	1,939	1,960	1,979
		$PS2_{RINS}$	0,752	1,025	**1,226**	**1,251**	**1,276**	**1,318**	**1,339**
35	2	$Modelo\,1$	1,132	2,250	6,667	13,029	25,331	60,074	114,173
		$PS1$	0,985	1,774	4,509	7,959	13,621	25,620	39,834
		$PS2$	0,935	1,706	4,107	6,675	10,454	18,153	25,903
		$PS2_{RINS}$	**0,875**	**1,461**	**2,935**	**4,131**	**5,700**	**7,727**	**8,906**
35	5	$Modelo\,1$	1,313	2,492	6,902	12,967	23,422	49,351	84,704
		$PS1$	1,247	2,226	5,213	8,534	13,363	22,962	33,170
		$PS2$	1,193	2,127	4,621	7,011	10,252	15,850	21,840
		$PS2_{RINS}$	**1,095**	**1,802**	**3,325**	**4,349**	**5,406**	**6,511**	**6,953**
35	10	$Modelo\,1$	0,926	1,492	3,018	4,516	6,629	10,090	12,678
		$PS1$	1,006	1,678	3,127	4,187	5,211	6,212	6,817
		$PS2$	0,949	1,584	2,910	3,788	4,499	5,121	5,449
		$PS2_{RINS}$	**0,917**	**1,449**	**2,266**	**2,526**	**2,743**	**3,012**	**3,207**
40	2	$Modelo\,1$	0,982	1,959	5,867	11,698	23,148	56,830	110,738
		$PS1$	0,858	1,586	4,151	7,402	12,662	24,315	37,616
		$PS2$	0,841	1,567	3,947	6,686	10,688	19,450	29,240
		$PS2_{RINS}$	**0,823**	**1,420**	**2,851**	**4,041**	**5,356**	**7,022**	**7,902**
40	5	$Modelo\,1$	1,604	3,166	9,224	18,026	34,872	82,371	154,301
		$PS1$	1,491	2,784	7,256	12,911	21,324	37,201	54,109
		$PS2$	1,441	2,674	6,761	10,969	16,472	25,952	35,063
		$PS2_{RINS}$	**1,412**	**2,519**	**5,346**	**7,412**	**9,497**	**12,074**	**13,650**
40	10	$Modelo\,1$	1,230	2,183	5,148	8,801	14,827	28,058	41,691
		$PS1$	1,270	2,329	5,066	7,527	10,428	15,250	18,897
		$PS2$	1,234	2,207	4,737	6,952	9,444	12,507	14,430
		$PS2_{RINS}$	**1,214**	**2,119**	**4,105**	**5,271**	**6,041**	**6,569**	**6,921**

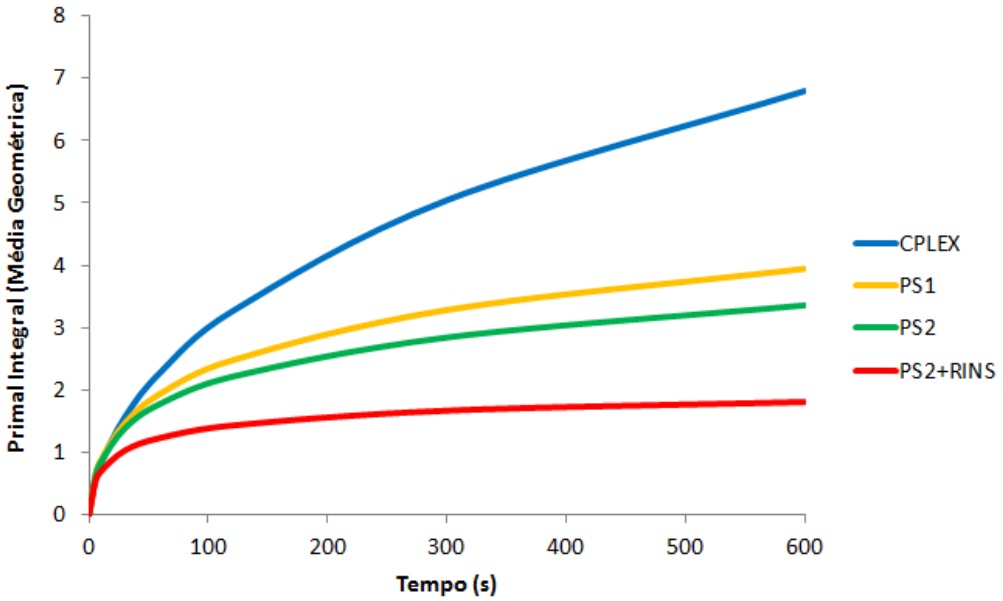


Figura 6.2. Gráfico da Média Geométrica usando Primal Integral.

De acordo com os experimentos realizados a partir das instâncias de médio porte, o $PS2_{RINS}$ obteve o melhor percentual de melhoramento (RPI) em relação a solução inicial fornecida pela heurística $HC7$, superando o Modelo 1 e as demais versões do PS. Também foi analisado o desempenho dos métodos em relação ao tempo gasto para obter cada melhoria. Segundo os resultados, o $PS2_{RINS}$ consegue obter as melhores soluções em poucos segundos de execução. No entanto, comparando o tempo médio para finalizar a execução de cada instância do problema, o $PS2$ conseguiu obter a menor média do tempo total de execução e encontrou o maior número de soluções ótimas.

Esses resultados mostram que, adicionando a heurística RINS ao $PS2$, melhores soluções são obtidas em poucos instantes de tempo, porém, não favoreceu na prova da otimalidade da solução. Em um trabalho futuro, é necessário avaliar o parâmetro utilizado pela heurística RINS que define o intervalo de nós da árvore de busca onde será aplicada a heurística. Neste trabalho foi adotado que a heurística RINS é aplicada em cada nó da árvore, o que aumentou o tempo gasto para finalizar a execução do modelo. Com essa parametrização é possível que melhore o desempenho do $PS2_{RINS}$ em relação ao tempo de execução.

Capítulo 7

Aplicação de Meta-heurísticas para o Problema $R_m \mid prec, S_{ijk} \mid C_{max}$

Neste capítulo será abordado o uso de duas meta-heurísticas para a resolução do problema proposto. Inicialmente, é discutido como o cálculo da função objetivo é realizado e como as soluções inviáveis são tratadas. Depois, são propostas duas buscas locais que serão utilizadas nas meta-heurísticas. Por fim, as meta-heurísticas implementadas são apresentadas e experimentos computacionais são realizados a fim de compará-las.

7.1 Representação da Solução

A solução do problema representa a ordem em que as tarefas são sequenciadas em cada máquina. Portanto, para representar uma solução é criado um vetor de m listas onde cada lista representa a sequência de tarefas em uma máquina. Cada lista contém todas as tarefas alocadas à máquina organizadas de acordo com a ordem de processamento. A Figura 7.1 ilustra a representação de uma solução do problema.

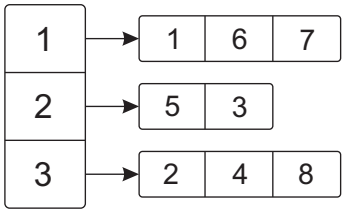


Figura 7.1. Representação da solução.

7.2 Avaliação de uma Solução

A solução é avaliada a partir do valor de *makespan*, que é o valor do tempo de conclusão da última tarefa a ser processada. Esse valor é obtido calculando o tempo de conclusão de todas as tarefas sequenciadas. Para problemas de sequenciamento em máquinas paralelas esse cálculo é simples, mas quando se considera restrição de precedência, o cálculo de torna um pouco mais complexo. Para determinar o tempo de conclusão de uma tarefa j, é necessário conhecer o tempo de conclusão de todas as tarefas predecessoras a j (definidas pela restrição de precedência) e o tempo de conclusão da tarefa k que é processada imediatamente antes da tarefa j na máquina i onde elas estão alocadas.

No Algoritmo 8 é apresentado um pseudocódigo do método *CalcularTempo*() usado para o cálculo do tempo de início I_j e do tempo de conclusão C_j de uma determinada tarefa $j \in J$. Para calcular o *makespan* de uma solução X, primeiro deve-se calcular o tempo de conclusão de cada tarefa j. Para isto, executa-se o Algoritmo *CalculaTempo()*, que tem como parâmetros de entrada a tarefa j e a máquina i onde a tarefa será processada. O algoritmo retornará o tempo de conclusão C_j. Então, o *makespan* será o maior C_j encontrado.

Algoritmo 8: CalcularTempo (j, i)

```
   saída: C_j;
 1 início
 2 |  k := tarefa processada imediatamente antes da tarefa j na máquina i;
 3 |  P := {k}∪ {todas as tarefas precessoras a tarefa j};
 4 |  para cada tarefa p ∈ P faça
 5 |  |  se C_p não é conhecido então
 6 |  |  |  q := máquina onde a tarefa p está alocada;
 7 |  |  |  CalcularTempo(p, q);
 8 |  |  fim
 9 |  fim
10 |  E := max{C_p | p ∈ P};
11 |  I_j := max{E, L_i + s_ijk};
12 |  C_j := I_j + p_ij;
13 |  L_i := C_j;
14 fim
```

Conforme as linhas 2 e 3, o conjunto P é formado pela tarefa k, antecessora imediata da tarefa j, e por todas as tarefas predecessoras da tarefa j. Nas linhas de 4 a 9, para cada tarefa p pertencente ao conjunto P é necessário calcular o seu

tempo de conclusão (C_p) caso ainda não tenha sido calculado. Esse calculo é feito chamando o Algoritmo *CalcularTempo*() recursivamente.

Somente quando o tempo de conclusão de todas as tarefas em P é conhecido é que o tempo de conclusão da tarefa j é calculado. Na linha 10, a variável E assume o maior tempo de conclusão das tarefas em P, cumprindo assim a restrição de precedência. Nas linhas 11 e 12, o tempo de início e o tempo de conclusão da tarefa j são calculados, respectivamente. O tempo de início I_j é o maior valor entre E e o tempo de conclusão da máquina i (L_i) somado ao tempo de preparação (s_{ijk}). O tempo de conclusão C_j é o tempo de início somado ao tempo de processamento p_{ij}. Após o cálculo do tempo de conclusão de cada tarefa da solução X, o valor do *makespan* será o maior tempo de conclusão obtido.

7.3 Buscas Locais

A busca local (BL) é uma estratégia de melhoria baseada em busca em vizinhança. Iterativamente a BL explora a vizinhança de uma solução corrente e seleciona-se, geralmente, a melhor solução vizinha. Se esta solução é melhor que a solução corrente, o processo é repetido até encontrar um ótimo local, ou seja, até que a solução corrente não possa ser melhorada. Nesta seção serão apresentadas duas buscas locais, denotadas de $BL1$ e $BL2$, que serão posteriormente utilizadas pelas meta-heurísticas desenvolvidas. Nas subseções a seguir será discutido o funcionamento de cada uma delas.

7.3.1 BL1

Nesta busca local, denotada por $BL1$, gera a vizinhança de uma dada solução X através de movimentos de inserção, que consiste em remover uma determinada tarefa t da solução X e simular a inserção de t em todas as posições possíveis da solução. Ao final, a solução atual é substituída pela inserção (solução vizinha) que apresentou o melhor valor para o *makepan*. No Algoritmo 9 é apresentado o pseudocódigo da heurística $BL1$.

Na linha 2 do algoritmo, o conjunto T é composto por todas as tarefas em ordem decrescente de acordo como o número de tarefas predecessoras de cada tarefa. Esse conjunto define a ordem com que as tarefas serão selecionadas para serem removidas e reinseridas em todas as posições. Portanto, a busca começa da tarefa com maior número de tarefas predecessoras.

Algoritmo 9: BL1 (X)

```
   saída: X;
 1 início
 2 |  T := {j_1, j_2, ..., j_n}: conjunto de n tarefas em ordem decrescente de
   |  acordo com o número de tarefas predecessoras de cada tarefa;
 3 |  for k = 1 to n do
 4 |  |  X_best := X_1 := X;
 5 |  |  X_1 := remove da solução X_1 a próxima tarefa j_k ∈ T;
 6 |  |  for i = 1 to m do
 7 |  |  |  p_inicial := primeira posição viável na máquina i para a tarefa j_k;
 8 |  |  |  p_final := última posição viável na máquina i para a tarefa j_k;
 9 |  |  |  enquanto p_inicial ≥ p_final faça
10 |  |  |  |  X_2 := inserir a tarefa j_k na posição p_inicial;
11 |  |  |  |  se C_max(X_2) < C_max(X_best) então
12 |  |  |  |  |  X_best := X_2;
13 |  |  |  |  p_inicial := p_inicial + 1;
14 |  |  |  fim
15 |  |  fim
16 |  |  se C_max(X_best) < C_max(X) então
17 |  |  |  X := X_best;
18 |  |  |  k := 1; // Reiniciar Busca
19 |  |  fim
20 |  fim
21 fim
```

Quando uma tarefa depende de muitas tarefas para ser inicializada, ou seja, possui muitas tarefas predecessoras, a tarefa tende a atrasar o seu inicio para aguardar as predecessoras serem finalizadas, gerando assim atrasos na máquina. Esse atraso gera ociosidade na máquina e provoca um aumento no valor do *makespan*. Devido a esta característica, a $BL1$ prioriza as tarefas com maior número de tarefas predecessoras a fim de minimizar a ociosidade provada pelas restrições de precedência e obter menor valor para o *makespan*.

A partir da linha 3, toda tarefa $j_k \in T$ é localizada e removida da solução X_1. Em seguida, a tarefa j_k é inserida em todas as posições viáveis de todas as máquinas da solução X_1. No entanto, uma inserção pode resultar em uma solução inviável, pois pode violar a restrição de precedência. Portanto, as posições viáveis são definidas antes das inserções serem feitas em cada máquina para evitar que gerem soluções inviáveis. Nas linhas 7 e 8 são definidas a posição inicial ($p_{inicial}$) e a posição final (p_{final}) de uma determinada máquina i na qual a tarefa j_k pode ser inserida. A análise realizada para definir as posições viáveis definidas pelas posições

inicial e final nas máquinas será discutida com mais detalhes na próxima seção. Nas linhas de 9 a 14 são realizadas as inserções nas posições viáveis da máquina.

De acordo com a linha 16, se houver melhora da solução atual X após todas as inserções realizadas pela tarefa j_k, a solução X é substituída pela melhor inserção realizada por j_k e a busca é reinicializada voltando à primeira tarefa do conjunto T. A busca local termina quando todas as tarefas em T foram reinseridas e nenhuma apresentou melhora.

7.3.2 BL2

Esta busca local, denotada por $BL2$, também é baseada na busca em vizinhança por inserção. O critério de seleção da tarefa a ser reinserida também é o mesmo usado pela $BL1$, ou seja, a busca local começa a partir da tarefa com maior número de tarefas predecessoras.

Uma característica presente em problemas com restrição de precedência é que tarefas com maior número de tarefas predecessoras tendem a serem alocadas próximas das últimas posições das máquinas. Visando melhorar o desempenho da heurística, nesta busca local a inserção é feita partindo da última posição viável de uma máquina para a primeira posição viável, ou seja, a inserção é feita de trás para frente em cada máquina. E para evitar inserções próximas das primeiras posições das máquinas, podendo assim gerar soluções vizinhas de baixa qualidade, foi criado o parâmetro *limit*. A inserção dentro de uma máquina i termina quando o número de inserções sem melhoras for igual ao número de tarefas alocadas na máquina i (γ) multiplicado pelo parâmetro *limit*.

No algoritmo 9 é apresentado o pseudocódigo da $BL2$. Na linha 2, o conjunto T armazena as tarefas em ordem decrescente de acordo com o número de tarefas predecessoras de cada tarefa k.

A partir da linha 3, cada tarefa $j_k \in T$ é localizada e removida da solução X_1. Em seguida, a tarefa j_k é inserida em todas as posições viáveis de todas as máquinas da solução X_1. Nas linhas 7 e 8 são definidos o intervalo inicial e final das posições viáveis em uma máquina i para uma determinada tarefa j_k.

Nas linhas de 10 a 19 são realizadas as inserções na máquina. As inserções começam sempre a partir da última posição viável (p_{final}) da máquina. Na linha 15, é verificado se o número de inserções sem melhora na máquina i alcança o valor do parâmetro *limit* multiplicado pelo número de tarefas alocadas na máquina i (γ). Caso esse valor seja atingido, a tarefa k é inserida a partir da próxima máquina $i + 1$.

Algoritmo 10: BL2 (X, $limit$)

```
   saída: X;
 1 início
 2 |  T := {j_1, j_2, ..., j_n}: conjunto de n tarefas em ordem decrescente de
   |  acordo com o número de tarefas predecessoras de cada tarefa;
 3 |  for k = 1 to n do
 4 |  |  X_best := X_1 := X;
 5 |  |  X_1 := remove da solução X_1 a próxima tarefa j_k ∈ T;
 6 |  |  for i = 1 to m do
 7 |  |  |  p_inicial := primeira posição viável na máquina i para a tarefa j_k;
 8 |  |  |  p_final := última posição viável na máquina i para a tarefa j_k;
 9 |  |  |  γ := número de tarefas alocadas na máquina i;
10 |  |  |  enquanto p_final ≥ p_inicial faça
11 |  |  |  |  X_2 := inserir a tarefa j_k na posição p_final;
12 |  |  |  |  se C_max(X_2) < C_max(X_best) então
13 |  |  |  |  |  X_best := X_2;
14 |  |  |  |  senão
15 |  |  |  |  |  se nº de inserções sem melhora ≥ (γ · limit) então
16 |  |  |  |  |  |  Pare; // Continuar na próxima máquina.
17 |  |  |  |  fim
18 |  |  |  |  p_final := p_final − 1;
19 |  |  |  fim
20 |  |  fim
21 |  |  se C_max(X_best) < C_max(X) então
22 |  |  |  X := X_best;
23 |  fim
24 fim
```

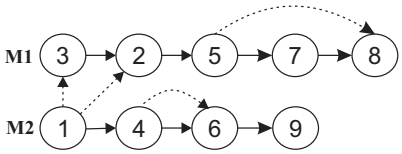
De acordo com a linha 21, se houver melhora da solução atual X após todas as inserções realizadas pela tarefa j_k, a solução X é substituída pela melhor inserção realizada por j_k e, na próxima iteração, a tarefa j_{k+1} é utilizada. A busca local termina quando todas as tarefas em T forem reinseridas. Diferentemente da $BL1$, a $BL2$ não é reinicializada após uma melhora, ou seja, todas as tarefas em T são reinseridas apenas uma vez. Devido a essa característica, a $BL2$ é mais rápida que a $BL1$.

7.4 Análise de Soluções Inviáveis

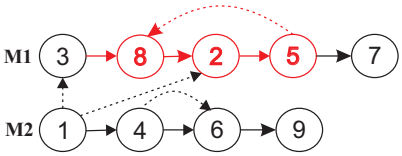
Em algumas heurísticas são aplicados movimentos na solução a fim de gerar soluções vizinhas. No entanto, esses movimentos podem afetar a restrição de precedência e então gerar soluções vizinhas inviáveis. Portanto, um método para evitar soluções

inviáveis é proposto.

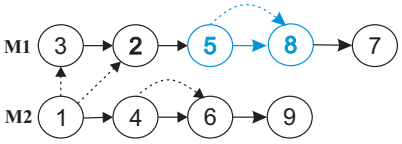
Considere um conjunto J contendo $n = 9$ tarefas, um conjunto M contendo $m = 2$ máquinas e também um conjunto $A = \{(1,2);(1,3);(5,8);(4,6)\}$ contendo as restrições de precedência entre as tarefas. Uma solução X (sequências de tarefas nas máquinas) e suas restrições de precedência são representadas pelo grafo de precedência ilustrado pela Figura 7.2(a).



(a) Grafo de Precedência.



(b) Inserção Inviável.



(c) Inserção Viável.

Figura 7.2. Exemplo de análise de soluções inviáveis.

As arestas tracejadas no grafo de precedência representam a relação de precedência entre duas tarefas, como por exemplo, a seta que liga a tarefa 5 à tarefa 8, simboliza que a tarefa 5 deve preceder a tarefa 8. As arestas solidas representam a ordem de alocação das tarefas em cada máquina.

Uma solução X se torna inviável se alguma tarefa iniciar antes de suas tarefas predecessoras, não respeitando assim as restrições de precedência. No grafo de precedência, uma solução X pode ser classificada como inviável se um ciclo é identificado no grafo. O objetivo é evitar que sejam realizados movimentos que gerem esses ciclos.

Nas buscas locais $BL1$ e $BL2$, uma tarefa j é removida da solução e reinserida em posições viáveis de cada máquina. No entanto, antes que a tarefa j comece a ser inserida em uma máquina, são verificadas as posições da máquina onde a inserção da tarefa não torne a solução inviável. São definidas duas posições: a posição inicial ($p_{inicial}$) e a posição final (p_{final}). As inserções são feitas entre as posições contidas

entre $p_{inicial}$ e p_{final}. Inserções fora deste intervalo não são realizadas, pois vão gerar inviabilidade.

Por exemplo, considere a solução X ilustrada pelo grafo de precedência da Figura 7.2(a), a tarefa 8 é removida da solução e reinserida entre as tarefas 3 e 2 da máquina M1, como mostra a Figura 7.2(b). No entanto, a solução torna-se inviável, por que um ciclo pode ser identificado entre as tarefas 5, 8 e 2. Neste caso, quaisquer inserção feita antes da tarefa 5 são inserções inviáveis, pois a tarefa 8 deve iniciar depois da tarefa 5. Portanto, a tarefa 8 somente pode ser inserida após a tarefa 5, como mostra a Figura 7.2(c). Neste caso, as posições da máquina M1 para alocar a tarefa 8 sem inviabilidade são as posições entre as tarefas 5 e 7. Se agora a restrição entre 5 e 8 for invertida, ou seja, se a tarefa 8 preceder a tarefa 5, a tarefa 8 somente pode ser inserida nas posições anteriores a tarefa 5.

7.5 Meta-heurísticas Propostas

7.5.1 GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma meta-heurística simples e de fácil implementação que iterativamente executa um procedimento de construção e um procedimento de busca local. Neste trabalho é proposto um algoritmo baseado no GRASP para resolver o problema abordado. Nas seções a seguir é apresentada a heurística constritiva e a estrutura do GRASP.

7.5.1.1 HC utilizada no GRASP

A fase construtiva do GRASP é um procedimento iterativo onde uma solução viável é construída inserindo um elemento por vez. A heurística construtiva do $GRASP$ proposta neste trabalho é denotada por HC_{GRASP}. O Algoritmo 11 apresenta um pseudocódigo do HC_{GRASP}.

O funcionamento do algoritmo é baseado nas heurísticas construtivas abordadas no Capítulo 5. Em cada iteração u do algoritmo, uma tarefa j é sequenciada. O algoritmo termina quando todas as tarefas tenham sido sequenciadas ($u = n$). Considerando t o menor tempo de conclusão de todas as máquinas, nas linhas de 4 a 7 é calculado o conjunto F, que contém todas as tarefas finalizadas até o instante de tempo t, e o conjunto D, que contém todas as tarefas disponíveis para sequenciamento no tempo t. Caso o conjunto D esteja vazio em t, os conjuntos F e

D são novamente calculados considerando um novo instante de tempo t, conforme calculado nas linhas de 8 a 13.

Na linha 14 é definida a lista de tarefas candidatas denominada LC. LC é composta por todas as tarefas contidas no conjunto D. Para cada tarefa em LC é calculado o menor tempo de conclusão para sequenciar cada tarefa j no final de cada máquina i da solução atual. Em seguida, as tarefas em LC são ordenadas crescentemente de acordo com o tempo de conclusão calculado.

Na linha 15 é definida a lista restrita de candidatos denominada LCR. A LCR é composta pelas tarefas melhor avaliadas em LC. O número de tarefas selecionadas de LC para compor a lista restrita é definida pelo parâmetro α da seguinte forma:

Algoritmo 11: HC_{GRASP}

```
   saída: C_max;
 1 início
 2     u ← 0, H ← ∅, L_i ← 0, ∀i ∈ M;
 3     enquanto u ≤ n faça
 4         T := {L_i | i ∈ M};
 5         t := min{L_i | L_i ∈ T};
 6         F := {j | C_j ≤ t, ∀j ∈ H};
 7         D := {j | ∀j ∈ (J − H), ∀k ∈ A_(k,j), k ∈ F};
 8         enquanto D = ∅ faça
 9             T := T \ {t};
10             t := min{L_i | L_i ∈ T};
11             F := {j | C_j ≤ t, ∀j ∈ H};
12             D := {j | ∀j ∈ (J − H), ∀k ∈ A_(k,j), k ∈ F};
13         fim
           /* selecionar uma tarefa j* ∈ D e uma máquina i* ∈ M   */
14         LC := CriarListaDeCandidatos(D);
15         LCR := DefinirListaRestrita(LC, α);
16         j* := Selecionar aleatoriamente uma tarefa j ∈ LCR;
17         i* := Selecionar aleatoriamente uma máquina i ∈ M;
           /* calcular tempo de conclusão e atualizar variáveis  */
18         z := max{C_k | ∀k ∈ A_(k,j*)};
19         C_j* := max{z, (L_i* + s_i*lj*)} + p_j*i*;
20         I_j* := C_j* − (s_i*lj* + p_j*i*);
21         L_i* := C_j*;
22         H := H ∪ {j*};
23         u := u + 1;
24     fim
25     C_max := max{C_j | j ∈ J};
26 fim
```

$|LC| \times \alpha$. Em seguida deve ser selecionada aleatoriamente uma tarefa $j^* \in LCR$ que deve ser alocada em uma máquina $i^* \in M$ também selecionada de forma aleatória.

Nas linhas de 18 a 23, o tempo de conclusão e o tempo de início da tarefa j^* são calculados e as demais variáveis também são atualizadas. Ao fim das iterações é obtida uma nova solução com o seu respectivo *makespan* (C_{max}).

7.5.1.2 Estrutura do Algoritmo GRASP

No algoritmo 12 é apresentado o pseudocódigo do GRASP. O GRASP vai executar iterativamente a fase construtiva e a fase de busca local até que o critério de parada seja satisfeito. Na linha 4, a fase construtiva é realizada através da heurística construtiva HC_{GRASP} utilizando o parâmetro α para definir o tamanho da lista LCR. Em seguida, na linha 5, a busca local $BL1$ é aplica sobre a solução X obtida pela fase construtiva. A melhor solução obtida nas iterações do GRASP (X_b) é retornada.

Algoritmo 12: GRASP

```
   saída: X_b;
 1 início
 2 |  C_max(X_b) := ∞;
 3 |  enquanto Critério de parada não satisfeito faça
 4 |  |  X := HC_GRASP(α);
 5 |  |  X' := BL1(X);
 6 |  |  se C_max(X') < C_max(X_b) então
 7 |  |  |  X_b := X';
 8 |  |  fim
 9 |  fim
10 fim
```

7.5.2 Iterated Greedy

A meta-heurística Iterated Greedy (IG) é simples e eficiente para resolver problemas de sequenciamento de tarefas (Ruiz & Stützle, 2007). Portanto, neste trabalho é proposto uma algoritmo baseado na meta-heurística IG para resolver o problema $R_m \mid prec, S_{ijk} \mid C_{max}$. Primeiro, uma solução inicial viável é gerada usando a heurística construtiva $HC7$ e depois a busca local $BL2$ é aplicada para melhorar a solução inicial. Então, o IG tenta melhorar iterativamente a solução atual através de três fases: a Destruição-Construção (DC), a busca local $BL2$ e critério de aceitação da solução corrente. Essas fases são executadas até que um critério de parada predefi-

nido tenha sido satisfeito. Nas próximas seções, a fase de Destruição-Construção e a estrutura do algoritmo IG são descritos com mais detalhes.

7.5.2.1 Fase de Destruição-Construção

A fase de Destruição-Construção, denotada por DC, consiste em duas etapas. Primeiro, a etapa de destruição remove aleatoriamente t diferentes tarefas da solução corrente. Para isso, uma máquina i é selecionada aleatoriamente, e então uma tarefa j alocada nesta máquina também é selecionada de forma aleatória. Esse processo é repetido t vezes. Todas as tarefas removidas são inseridas no conjunto R e a solução resultante é denominada de solução parcial X_r.

Durante a etapa de construção, uma tarefa j é aleatoriamente selecionada do conjunto R e reinserida em todas as posições da solução X_r considerando que essas inserções não podem gerar soluções inviáveis. Dentre as posições inseridas onde a solução é viável, a tarefa j é alocada na posição que resultar no menor valor de *makespan* para X_r. Esse processo é repetido até que o conjunto R fique vazio, ou seja, quando todas as tarefas de R estejam reinseridas em X_r, obtendo assim uma nova solução completa.

Na implementação do IG é considerado o parâmetro t que tem como objetivo controlar o nível de destruição realizada sobre a solução, determinando a quantidade de soluções a serem removidas. O valor deste parâmetro é incrementado em um $(t = t + 1)$ toda vez que a solução corrente não melhora após a aplicação de uma busca local. O valor de t volta a valer 1 caso a solução tenha sido melhorada ou quando atinge um valor máximo, denominado de nível máximo de destruição d_{max}.

7.5.2.2 Estrutura do Iterated Greedy

Um pseudocódigo descrevendo o IG é apresentado no Algoritmo 13. O IG tem dois parâmetros de entrada: o parâmetro d_{max}, que define o nível máximo de destruição, e o parâmetro $limit$, que restringe o número de inserções sem melhora na busca local $BL2$. A $BL2$ é utilizada no IG por ser uma busca local mais rápida que a $BL1$, possibilitando que o IG realize um número maior de iterações.

Nas linhas 2 e 3, a solução inicial é obtida pela heurística construtiva $HC7$ e então melhorada pela busca local $BL2$. Na linha 4, o nível de destruição é inicializado com $t = 1$. As iterações do IG são computados nas linhas de 6 a 21 e executam até que o critério de parada seja satisfeito. Durante cada iteração, a solução corrente X é submetida a fase DC e melhorada pela busca local $BL2$, resultando na solução X'. Se a solução X' for melhor que a solução corrente X, o nível de destruição t

é atualizado para 1 ($t = 1$) e $X = X'$. Caso contrário, o nível de destruição t é incrementado ($t = t + 1$). O valor máximo que t pode assumir é limitado por d_{max}.

Em seguida é verificado o critério de aceitação. Se a solução X' é aceita pelo critério, a solução corrente X é substituída pela solução X' mesmo sendo uma solução pior. O critério de aceitação adotado, aplicado na linha 18, é o mesmo aplicado por Ruiz & Stützle (2007), que utiliza uma constante denominada temperatura (T). A temperatura usada é semelhante a temperatura usada pela meta-heurística *Simulated Annealing*, inicialmente proposta por Osman & Potts (1989). Basicamente, a temperatura é calculada por $T = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}/(n \cdot m \cdot 10)$, onde p_{ij} é o tempo de processamento da tarefa j na máquina i, n é o número de tarefas e m é o número de máquinas.

Algoritmo 13: Iterated Greedy Algorithm (IG)

```
   saída: X_b;
 1 início
 2 |  X := HC7();
 3 |  X := BL2(X, limit);
 4 |  t := 1;
 5 |  X_b := X;
 6 |  enquanto critério de parada não satisfeito faça
 7 |  |  X' := Destruction-Construction (X, t);
 8 |  |  X' := BL2(X', limit);
 9 |  |  se C_max(X') < C_max(X) então
10 |  |  |  t := 1;
11 |  |  |  X := X';
12 |  |  |  se C_max(X') < C_max(X_b) então
13 |  |  |  |  X_b := X';
14 |  |  senão
15 |  |  |  t := t + 1;
16 |  |  |  se t > d_max então
17 |  |  |  |  t := 1;
18 |  |  |  se rand(0,1) ≤ exp(−(C_max(X') − C_max(X_b))/T) então
19 |  |  |  |  X := X';
20 |  |  fim
21 |  fim
22 fim
```

7.6 Experimentos Computacionais

Nesta seção são apresentados os experimentos computacionais realizados a fim de analisar e comparar o desempenho das meta-heurísticas GRASP e IG implementadas para resolver o problema $R_m \mid prec, S_{ijk} \mid C_{max}$. O GRASP e o IG foram implementados na linguagem C++ e no ambiente de desenvolvimento Microsoft Visual C++ 2010. A configuração do computador utilizado nos experimentos é o mesmo utilizado pelos os experimentos apresentados nos capítulos anteriores.

Para estes experimentos computacionais foi utilizado o conjunto de instâncias de grande porte. Para avaliar os resultados obtidos é calculado o RPD (Desvio Percentual Relativo) para a solução obtida em cada uma das instâncias. O RPD é definido pela seguinte equação:

$$RPD(\%) = \frac{f - f_{best}}{f_{best}} \times 100 \tag{7.1}$$

onde f_{best} é o valor da função objetivo da melhor solução conhecida para uma determinada instância do problema e f é a média da função objetivo obtida após 5 execuções da heurística em questão.

Também foi calculado o RPI (*Relative Percentage Improve*) de todas as instâncias. Essa métrica é usada para avaliar a porcentagem de melhoria provocada pela heurística em relação à solução inicial. A métrica RPI pode ser definida como:

$$RPI(\%) = \frac{f_{s.i.} - f_{heuristica}}{f_{s.i.}} \times 100, \tag{7.2}$$

onde $f_{s.i.}$ é o valor da função objetivo da solução inicial fornecida pela heurística $HC7$, e $f_{heuristica}$ é a média da função objetivo obtida após 5 execuções da heurística implementada.

O critério de parada utilizado pelo GRASP e pelo IG nos experimentos é baseado na quantidade de tempo de CPU, cujo tempo é calculado da seguinte forma: $tempo = n \times log_2 m \times 100$ milissegundos.

Antes de apresentar a comparação dos resultados obtidos pelo GRASP e IG, apresentam-se, nas seções seguintes, a calibração dos parâmetros desses algoritmos.

7.6.1 Calibração dos Parâmetros do GRASP

Antes de resolver todas as instâncias do problema com o algoritmo GRASP, faz-se necessário a calibração do parâmetro α, que determina o nível de aleatoriedade no processo de escolha das tarefas contidas na lista restrita de candidatos. Foi realizado

uma Análise de Variância (ANOVA) para determinar o melhor valor do parâmetro α. Para o parâmetro α os seguintes valores foram testados: $\alpha = \{0.2, 0.4, 0.6, 0.8, 1\}$, resultando em 4 configurações do algoritmo.

Foram geradas para a calibração 135 instâncias de grande porte. Para cada instância, o algoritmo GRASP foi executado 5 vezes. Os resultados obtidos são analisados utilizando o valor do RPD(%) calculado para cada uma das instâncias.

As três principais hipóteses da ANOVA (normalidade, igualdade de variância e independência dos resíduos) foram verificadas. A Figura 7.4 mostra o gráfico de intervalos HSD de *Tukey* com nível de confiança de 95% usada para comparar o desempenho do algoritmo para cada valor de α. Na figura pode ser visto que o GRASP obtém melhores resultados com $\alpha = 0.2$.

Embora que, para $\alpha = 0.2$ e $\alpha = 0.4$ os resultados não sejam estatisticamente diferentes, com $\alpha = 0.2$ obtem-se a melhor média. Nos próximos experimentos com o GRASP será utilizado $\alpha = 0.2$.

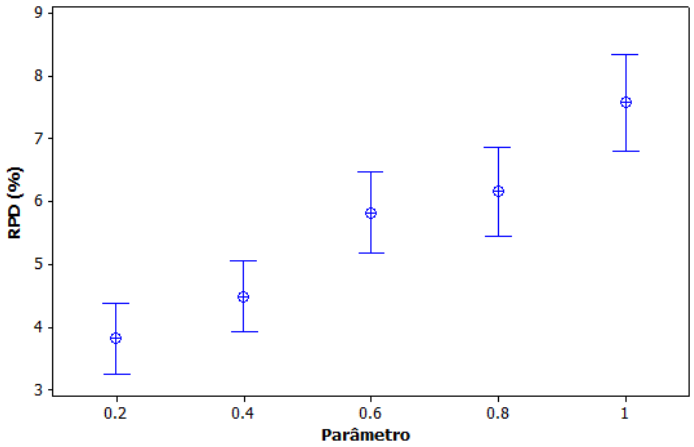


Figura 7.3. Gráfico de Intervalos e Médias para Calibração do Parâmetro α.

7.6.2 Calibração dos Parâmetros do IG

A fim de determinar o melhor valor para os parâmetros do IG, são realizados testes computacionais preliminares a fim de calibrar os seguintes parâmetros: o nível máximo de destruição (d_{max}), usado na fase de *Destruição-Construção*, e o parâmetro *limit*, usado na fase de busca local do IG. Para esses parâmetros, os seguintes valores foram testados: $d_{max} \in \{n/5, n/6, n/7, n/8, n/9\}$ e $limit \in \{20\%, 40\%, 60\%, 80\%, 100\%\}$, resultando em 25 configurações combinando os dois parâmetros para o algoritmo IG.

Para realizar os testes, foram utilizadas 135 instâncias de grande porte. Os resultados obtidos são avaliados usando a média do RPD(%) calculado para cada uma das instâncias. Cada instância foi executada 5 vezes. Os resultados também são analisados por meio do teste ANOVA.

As três principais hipóteses da ANOVA (normalidade, igualdade de variância e independência dos resíduos) foram verificadas. A Figura 7.4 mostra o gráfico de intervalos HSD de *Tukey* com nível de confiança de 95% usada para comparar o desempenho de cada configuração do IG. Na figura pode ser visto que não existem diferenças significativas para os diferentes valores dos dois parâmetros, no entanto, o IG obtém melhores médias com $d_{max} = n/7$ e $limit = 60\%$.

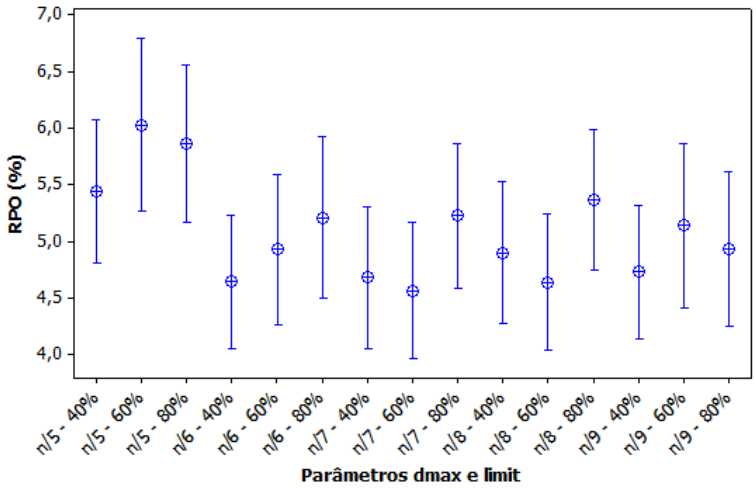


Figura 7.4. Gráfico de Intervalos e Médias para Calibração do Parâmetro d_{max} e $limit$.

7.6.3 Experimento 1: Testes com Instâncias de Médio Porte

No primeiro experimento foi utilizado o conjunto de instâncias de médio porte. O desempenho do Modelo 1, do GRASP e do IG foram comparados em relação a média do RPD(%) e a média do tempo de execução de cada instância. O CPLEX foi utilizado para resolver o Modelo 1 com um limite máximo de 3600 segundos de tempo de CPU para cada uma das instâncias. Se o tempo limite é atingido e nenhuma solução ótima é encontrada, a melhor solução viável obtida é retornada pelo CPLEX.

Na Tabela 7.1 são apresentados os resultados obtidos pelo Modelo 1, pelo GRASP e pelo IG. Para cada configuração $(n \times m)$ com 45 problemas, a tabela

mostra o RPD médio para cada heurística comparada.

Pode ser observado que o IG obtém melhores valores de RPD em 10 de 12 grupos de instâncias. Analisando a média total do RPD, o Modelo 1 obteve 6,77%, o GRASP 8,60%, e o IG 0,09%.

Tabela 7.1. Desempenho das Heurísticas GRASP e IG para o Conjunto de Instâncias de Médio Porte

		Modelo 1		**GRASP**		**IG**	
n	m	RPD(%)	Tempo	RPD(%)	Tempo	RPD(%)	Tempo
15	2	0,36	1509,13	7,92	1,50	**0,00**	1,50
15	5	**0,00**	17,74	7,78	3,48	0,05	3,48
15	10	**0,00**	7,03	9,79	4,98	**0,00**	4,98
20	2	3,08	2859,41	6,35	2,00	**0,00**	2,00
20	5	3,00	1295,67	10,37	4,64	**0,08**	4,64
20	10	**0,00**	196,63	7,13	6,64	0,36	6,64
25	2	6,46	3275,01	4,92	2,50	**0,00**	2,50
25	5	8,26	2601,29	12,08	5,80	**0,13**	5,80
25	10	4,35	971,93	11,27	8,30	**0,44**	8,30
30	2	13,23	3563,18	6,07	3,00	**0,05**	3,00
30	5	22,35	3317,94	8,95	6,97	**0,00**	6,97
30	10	20,21	2248,84	10,61	9,97	**0,00**	9,97
Média:		6,77	1821,98	8,60	4,98	**0,09**	4,98

Outra importante informação a respeito desse experimento é que do total de 540 instâncias, o IG foi superior ao Modelo 1 em 215 instâncias, o Modelo 1 foi melhor que o IG em 63 instâncias e o IG e o Modelo 1 obtiveram os mesmos resultados em 262 instâncias. O Modelo 1 foi capaz de obter 291 soluções ótimas de 540 instâncias. Além disso, o tempo médio gasto pelo IG (4,98 s) é aproximadamente 374 vezes menor que o gasto pelo Modelo 1 (1863,43 s).

7.6.4 Experimento 2: Testes com Instâncias Grandes

Neste experimento é comparado o desempenho entre as heurísticas propostas neste capítulo utilizando o conjunto de instâncias de grande porte. A heurística construtiva $HC7$ discutida no Capítulo 5 é executada para todas as instâncias. Cada solução obtida por $HC7$ é fornecida como solução inicial das buscas locais $BL1$ e $BL2$ que também foram executadas, sendo que o parâmetro $limit$ da $BL2$ é definido com valor igual a 60%. Já o GRASP foi executado com o parâmetro $\alpha = 0.2$ e o IG com os parâmetros $d_{max} = n/7$ e $limit = 60\%$. O GRASP e o IG foram submetidos aos mesmos critérios de parada conforme definido no início da seção 7.6.

Na Tabela 7.2 são apresentados os resultados obtidos pela $BL1$, $BL2$, GRASP e IG. Para cada configuração $(n \times m)$ com 45 problemas, a tabela mostra o RPI(%)

médio e o tempo de execução médio em segundos de cada heurística. O RPI representa a porcentagem de melhoramento obtida pela heurística em relação a heurística construtiva $HC7$.

Tabela 7.2. Porcentagem de Melhoramento e Tempo de Execução das Heurísticas Propostas

		***BL*1**		***BL*2**		**GRASP**		**IG**	
n	m	RPI(%)	Tempo	RPI(%)	Tempo	RPI(%)	Tempo	RPI(%)	Tempo
60	5	11,33	0,08	7,78	0,02	21,13	13,93	**27,56**	13,93
60	10	10,01	0,07	6,55	0,02	17,76	19,93	**25,73**	19,93
60	15	7,91	0,07	5,34	0,03	15,33	23,44	**22,32**	23,44
90	5	8,29	0,23	5,25	0,04	17,65	20,90	**25,75**	20,90
90	10	8,02	0,21	5,72	0,06	17,87	29,90	**26,14**	29,90
90	15	6,54	0,21	4,48	0,07	15,03	35,16	**22,64**	35,16
120	5	8,69	0,55	5,31	0,09	16,37	27,86	**24,75**	27,86
120	10	6,49	0,41	4,96	0,12	16,42	39,86	**23,93**	39,86
120	15	6,39	0,43	4,64	0,16	15,15	46,88	**19,46**	46,88
150	5	7,66	0,86	5,66	0,17	14,01	34,83	**22,46**	34,83
150	10	6,76	0,90	4,74	0,24	13,32	49,83	**21,69**	49,83
150	15	5,47	0,75	4,20	0,27	13,47	58,60	**18,63**	58,60
180	5	4,49	0,30	7,81	1,75	12,34	41,79	**20,52**	41,79
180	10	5,41	1,20	3,80	0,41	11,32	59,79	**19,63**	59,79
180	15	4,78	1,17	3,72	0,48	10,74	70,32	**14,81**	70,32
Médias:		7,43	0,59	5,11	0,17	15,19	38,20	**22,40**	38,20

Como pode ser observado, a $BL1$ obteve RPI médio igual a 7,43, enquanto a $BL2$ obteve média igual a 5,11. A $BL1$ obteve maior percentual de melhoramento, porém, gasta-se mais tempo de execução que a $BL2$. A $BL2$ gasta aproximadamente 3,5 vezes menos tempo de execução em média.

Comparando agora o GRASP e o IG, o IG conseguiu uma maior porcentagem de melhoramento (RPI) para todas as instâncias, totalizando um RPI médio igual a 22,40% contra 15,19% do GRASP.

A Figura 7.5 apresenta o gráfico de intervalos HSD de *Tukey* com nível de confiança de 95% utilizado para comparar o desempenho entre o GRASP e o IG. Na figura pode ser visto que o IG possui RPD médio significativamente menor que o GRASP (quanto menor, melhor).

Analisando os resultados apresentados, pode-se concluir que o IG obteve o melhor desempenho que o GRASP em relação a qualidade das soluções encontradas.

7.6.5 Experimento 3: Comparação entre o PS e o IG

Este experimento tem como objetivo comparar o desempenho do PS, apresentado no Capítulo 6, com algoritmo IG que é a heurística que apresentou os melhores

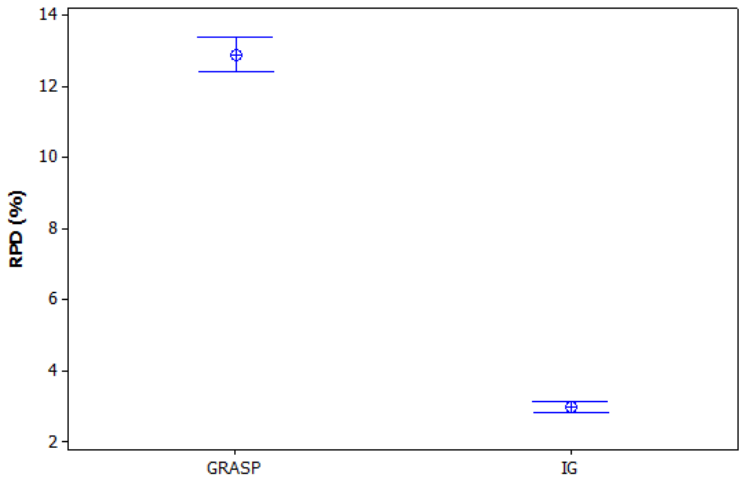


Figura 7.5. Gráfico de Intervalos e Médias do RPD (%) do GRASP e IG.

resultados. Neste experimento será utilizado o conjunto de instâncias de médio porte. Todas as versões do PS e o IG são inicializados com a heurística construtiva $HC7$ e limitados a 600 segundos de tempo de execução. Os parâmetros utilizados pelo PS e IG são os mesmos definidos anteriormente: $\theta = 1$ para o PS e $d_{max} = n/7$ e $limit = 60\%$ para o IG.

Na primeira parte da Tabela 7.3 são apresentados os valores médios de RPI(%) para o Modelo 1, $PS1$, $PS2$, $PS2_{RINS}$ e IG que mede o percentual de melhoria em relação a solução inicial gerada pela heurística $HC7$. Como pode ser observado, o IG apresentou o melhor percentual de melhoria, superando levemente os resultados do $PS2_{RINS}$. Na segunda parte da tabela, são apresentados os tempos de execução médio para cada grupo de instâncias. Como o IG é uma heurística de aproximação, não é possível provar a otimalidade da solução encontrada, portanto, o algoritmo utiliza todo o tempo limite de 600 segundos. O $PS2$ possui o menor tempo médio de execução (319,81 s) e encontra o maior número de soluções ótimas (386 soluções).

Na Figura 7.6 são apresentados, para seis instâncias distintas, um gráfico que demostra a atualização da melhor solução obtida ao decorrer do tempo de execução dos algoritmos. Para isso, a cada melhora obtida por um algoritmo foi armazenado o tempo gasto para encontrar a melhor solução e, ao final, foi gerado o gráfico $tempo(s) \times C_{max}(x)$ para as seis instâncias distintas. Como pode ser observado, o IG conseguiu um bom desempenho, obtendo a melhor solução em poucos instantes de tempo. A diferença de desempenho fica mais evidente nas instâncias das Figuras 7.6(d), 7.6(e) e 7.6(f).

A métrica *Primal Integral* apresentada no Capítulo 6 foi calculada para com-

Tabela 7.3. RPI(%) e tempo médio (s) gasto pelo $PS1$, $PS2$, $PS2_{RINS}$ e IG.

n	m	RPI (%)				Tempo (s)			
		$PS1$	$PS2$	$PS2_{RINS}$	IG	$PS1$	$PS2$	$PS2_{RINS}$	IG
15	2	20,11	20,03	19,92	**20,19**	261,16	149,98	499,82	600,00
15	5	**17,38**	17,15	**17,38**	**17,38**	1,85	1,86	29,25	600,00
15	10	**14,61**	14,59	**14,61**	**14,61**	1,30	1,09	0,99	600,00
20	2	17,07	17,44	17,60	**18,73**	486,68	481,16	583,10	600,00
20	5	21,16	21,13	21,10	**21,32**	81,16	40,59	331,49	600,00
20	10	**18,05**	**18,05**	**18,05**	**18,05**	4,05	3,89	16,42	600,00
25	2	17,47	17,37	18,68	**20,15**	562,43	534,86	591,47	600,00
25	5	22,60	22,67	23,73	**24,16**	321,90	291,01	509,72	600,00
25	10	**19,66**	19,63	**19,66**	19,50	12,91	11,90	134,22	600,00
30	2	15,67	16,81	18,22	**22,33**	600,00	600,00	600,00	600,00
30	5	21,16	22,31	24,58	**26,37**	537,55	526,95	580,44	600,00
30	10	**21,34**	21,28	21,31	21,22	87,98	79,59	357,97	600,00
35	2	14,84	16,72	19,05	**23,87**	600,00	600,00	600,00	600,00
35	5	17,64	19,53	23,25	**25,63**	563,72	548,27	586,81	600,00
35	10	20,06	20,51	20,69	**20,77**	287,68	274,34	494,26	600,00
40	2	12,53	13,76	17,30	**24,03**	600,00	600,00	600,00	600,00
40	5	19,07	21,91	25,07	**30,42**	579,42	576,60	600,00	600,00
40	10	19,42	20,97	22,60	**23,20**	441,50	433,78	534,00	600,00
Médias:		18,32	18,99	20,16	**21,77**	335,12	319,81	425,00	600,00

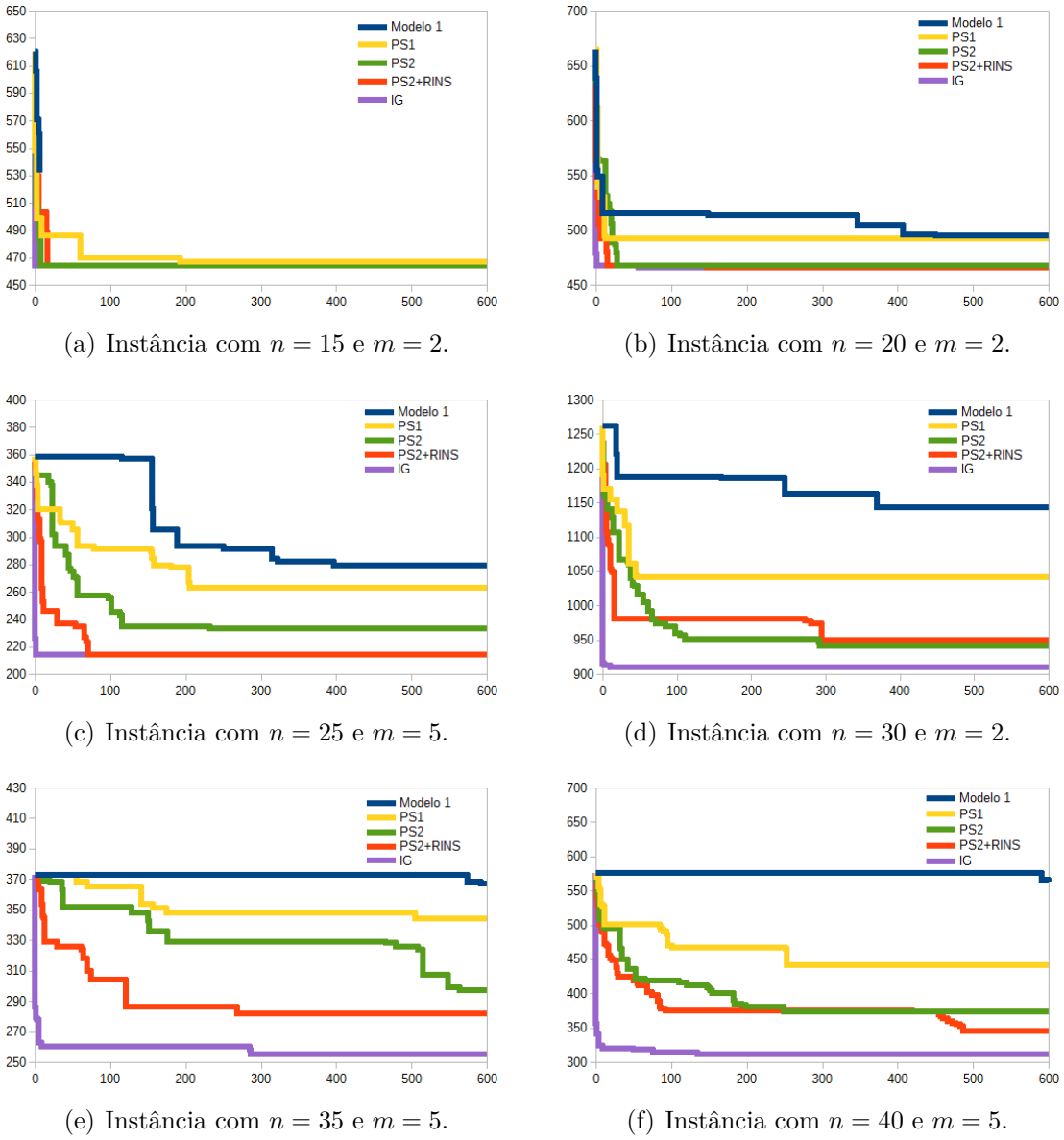
parar o desempenho do IG com as versões do PS em relação ao tempo gasto para obter cada solução melhorada. Nos intervalos de tempo definidos em 5, 10, 30, 60, 120, 300 e 600 segundos é calculada a integral para cada instância do problema e para todos os algoritmos implementados. Quanto menor é o valor da integral em cada intervalo de tempo, melhor é o desempenho do algoritmo.

A Tabela 7.4 apresenta a média geométrica da integral $P(t)$ para cada método aplicado e em cada intervalo de tempo. Como pode ser observado, o IG obteve os menores valores de integral em todos os intervalos de tempo.

Tabela 7.4. Média Geométrica do Primal Integral depois de 5, 10, 30, ..., 600 segundos (Quanto Menor Melhor).

Tempo (s)	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$	IG
5	0,584691	0,661665	0,639025	0,561078	**0,022888**
10	0,903359	0,971179	0,940414	0,784435	**0,026734**
30	1,710422	1,646756	1,564943	1,201258	**0,032795**
60	2,499431	2,205695	2,047363	1,495901	**0,036408**
120	3,571091	2,859562	2,607156	1,835624	**0,039469**
300	5,539683	3,919548	3,510906	2,380035	**0,042695**
600	7,496294	4,883194	4,363725	2,891475	**0,044173**

Na Figura 7.7 é apresentado o gráfico da média geométrica da integral de cada algoritmo implementado. De acordo com o gráfico, o Modelo 1 possui uma curva



(a) Instância com $n = 15$ e $m = 2$.

(b) Instância com $n = 20$ e $m = 2$.

(c) Instância com $n = 25$ e $m = 5$.

(d) Instância com $n = 30$ e $m = 2$.

(e) Instância com $n = 35$ e $m = 5$.

(f) Instância com $n = 40$ e $m = 5$.

Figura 7.6. Atualização da solução de diferentes instâncias ao decorrer de 600 segundos ($Tempo(s) \times C_{max}(x)$) usando o Modelo 1, $PS1$, $PS2$, $PS2_{RINS}$ e IG.

mais aberta, ou seja, possui os maiores valores para a média geométrica da integral. As versões do PS apresentaram uma curva mais acentuada, possuindo resultados melhores que o Modelo 1. Já o IG possui praticamente uma reta muito próxima de zero, gerando uma diferença significativa em relação as versões do PS e o Modelo 1. Esse resultado mostra a eficiência do IG em encontrar as melhores soluções do problema logo nos primeiros instantes de tempo de execução.

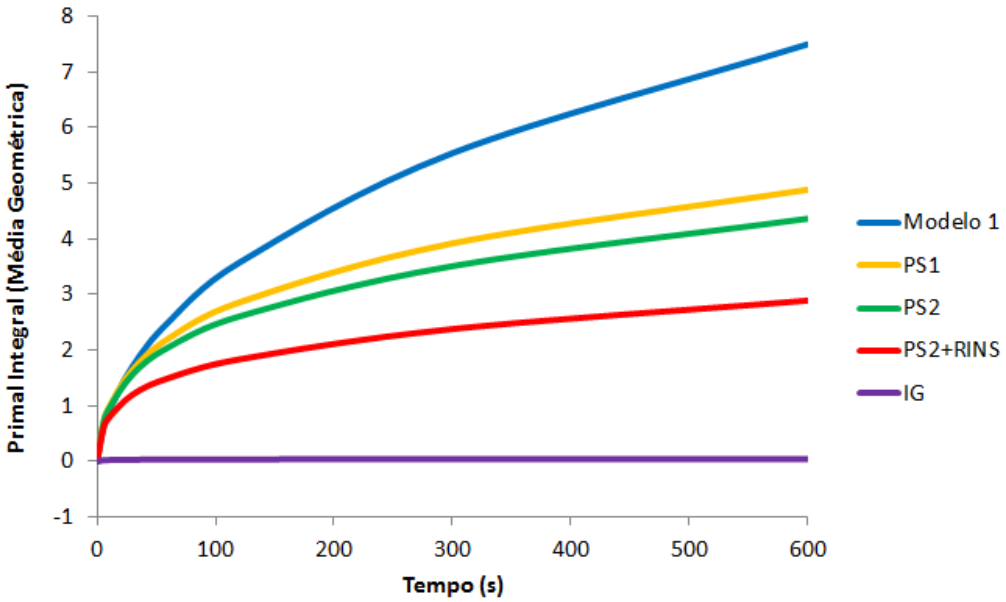


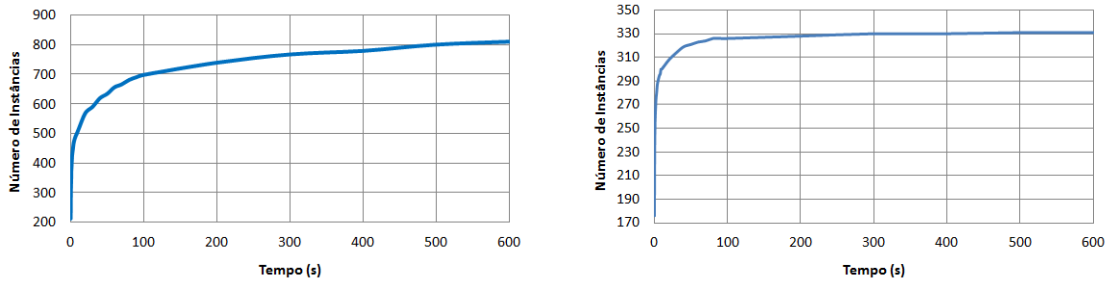
Figura 7.7. Gráfico da Média Geométrica usando Primal Integral.

7.6.6 Experimento 4: Usando o IG como Solução Inicial do PS

O objetivo deste experimento é avaliar o comportamento do PS a partir de uma solução inicial de alta qualidade obtida através de algum tempo de execução do IG. Neste experimento é considerado o conjunto de instâncias de médio porte.

Primeiramente, foi necessário definir o tempo médio em que o IG gasta para atingir a melhor solução de cada instância. Para isso, foram gerados os dois gráficos apresentados na Figura 7.8. O primeiro gráfico apresenta, em determinados tempos de execução, a quantidade de instâncias onde o IG encontrou a sua melhor solução. Já o segundo gráfico apresenta a quantidade de instâncias por instante de tempo onde o IG encontrou a solução ótima considerando apenas as instâncias onde a otimalidade é conhecida. Pode-se notar que após 100 segundos os dois gráficos passam a ficar menos acentuados. Portanto, o IG consegue encontrar as melhores soluções ou as soluções ótimas conhecidas em até 100 segundos em média. Este tempo será o tempo limite de execução para o IG. Após esse tempo, a solução obtida é fornecida como solução inicial para as versões do PS e para o CPLEX na resolução do Modelo 1. Por sua vez, o PS e o Modelo 1 continuam a execução até atingir no máximo 500 segundos, totalizando assim 600 segundos de execução, ou até que a otimalidade seja provada.

A Tabela 7.5 apresenta os valores médios de RPD(%) de cada grupo de instâncias para o Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$. Na primeira parte da tabela,



(a) Tempo x Nº de Instâncias com a Melhor Solução.

(b) Tempo x Nº de Instâncias com a Solução Ótima.

Figura 7.8. Gráfico contendo o número de instâncias que atingiram a melhor solução ou a solução ótima ao decorrer de 600 segundos.

os métodos utilizam a heurística $HC7$ como solução inicial e, na segunda parte, utilizam a meta-heurística IG com 100 segundos de execução. Como pode ser observado, os métodos implementados utilizando IG conseguiram os menores valores de RPD, significando que utilizando o IG como solução inicial conseguiram encontrar as melhores soluções.

Tabela 7.5. RPD(%) médio para o Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$ com HC7 e IG como solução inicial.

		$HC7 + método$				$IG + método$			
n	m	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$
15	2	0,77	0,10	0,21	0,35	**0,00**	**0,00**	**0,00**	**0,00**
15	5	0,00	0,00	0,32	0,00	**0,00**	**0,00**	**0,00**	**0,00**
15	10	0,00	0,00	0,02	0,00	**0,00**	**0,00**	**0,00**	**0,00**
20	2	3,67	2,18	1,66	1,46	**0,00**	**0,00**	**0,00**	**0,00**
20	5	0,48	0,25	0,27	0,33	**0,00**	**0,01**	**0,01**	**0,01**
20	10	0,00	0,00	0,00	0,00	**0,00**	**0,00**	**0,00**	**0,00**
25	2	10,74	3,57	3,69	1,90	**0,01**	**0,01**	**0,01**	**0,01**
25	5	4,71	2,39	2,21	0,56	0,04	0,06	**0,04**	0,06
25	10	0,00	0,00	0,04	0,00	**0,00**	**0,00**	0,07	**0,00**
30	2	20,30	8,79	7,31	5,40	**0,00**	**0,00**	**0,00**	**0,00**
30	5	21,31	7,66	5,90	2,50	**0,00**	**0,00**	**0,00**	0,03
30	10	1,03	0,00	0,09	0,03	**0,00**	**0,00**	**0,00**	0,21
35	2	26,99	12,22	9,52	6,41	**0,00**	**0,00**	**0,00**	**0,00**
35	5	27,12	11,35	8,72	3,21	0,08	0,07	0,06	**0,05**
35	10	7,22	1,20	0,60	0,32	0,41	**0,06**	0,25	0,25
40	2	30,04	15,45	13,75	8,87	0,01	0,01	0,01	**0,00**
40	5	39,35	16,87	12,56	7,80	0,04	0,04	0,03	**0,01**
40	10	18,62	5,75	3,43	1,02	0,64	0,49	0,42	**0,32**
Médias		11,80	4,88	3,91	2,23	0,07	**0,04**	0,05	0,05

No entanto, como mostra na primeira parte da Tabela 7.6, os métodos conseguiram um valor de RPI(%) maior utilizando a heurística $HC7$ como solução inicial. Na segunda parte é mostrado o RPI para os métodos implementados utilizando o

IG como solução inicial. Tanto o modelo quanto as versões do PS apresentaram um baixo valor de RPI e em muitas instâncias apresentaram RPI = 0%, ou seja, melhoram muito pouco ou não melhoraram a solução inicial fornecida pelo IG.

Na Tabela 7.7 é apresentado o tempo médio (em segundos) gasto por cada método implementado. Na primeira parte da tabela, é computado o tempo de execução da heurística $HC7$ mais o método com limite de 600 segundos. Na segunda parte, é computado o tempo gasto pelo IG limitado a 100 segundos somado ao tempo gasto pelo método limitado a 500 segundos. Como pode ser observado, não houve melhora em relação ao tempo de execução total utilizando o IG como solução inicial. Se for desprezado os 100 segundos gasto pelo IG, o método $PS2$ com IG como solução inicial obtém tempo médio total de 257,46 segundos, sendo inferior ao tempo gasto pelo $PS2$ utilizando a heurística $HC7$.

Tabela 7.6. RPI(%) médio para o Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$ com HC7 e IG (100 s) como solução inicial.

		HC7 + método				*IG + método*			
n	m	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$
15	2	19,60	**20,11**	20,03	19,92	0,00	0,00	0,00	0,00
15	5	**17,38**	**17,38**	17,15	**17,38**	0,00	0,00	0,00	0,00
15	10	**14,61**	**14,61**	14,59	**14,61**	0,00	0,00	0,00	0,00
20	2	15,89	17,07	17,44	**17,60**	0,00	0,00	0,00	0,00
20	5	20,98	**21,16**	21,13	21,10	0,01	0,00	0,00	0,00
20	10	**18,05**	**18,05**	**18,05**	**18,05**	0,00	0,00	0,00	0,00
25	2	12,12	17,47	17,37	**18,68**	0,00	0,00	0,00	0,00
25	5	20,98	22,60	22,67	**23,73**	0,03	0,01	0,03	0,01
25	10	**19,66**	**19,66**	19,63	**19,66**	0,24	0,24	0,17	0,24
30	2	7,07	15,67	16,81	**18,22**	0,00	0,00	0,00	0,00
30	5	12,02	21,16	22,31	**24,58**	0,03	0,03	0,03	0,00
30	10	20,59	21,34	21,28	**21,31**	0,28	0,28	0,28	0,08
35	2	3,98	14,84	16,72	**19,05**	0,00	0,00	0,00	0,00
35	5	6,90	17,64	19,53	**23,25**	0,00	0,01	0,02	0,03
35	10	15,85	20,06	20,51	**20,69**	0,30	0,63	0,45	0,45
40	2	1,77	12,53	13,76	**17,30**	0,00	0,00	0,00	0,01
40	5	4,13	19,07	21,91	**25,07**	0,00	0,00	0,01	0,03
40	10	10,56	19,42	20,97	**22,60**	0,21	0,36	0,43	0,52
Médias		13,45	18,32	18,99	**20,16**	0,06	0,09	0,08	0,08

Analisando a quantidade de soluções ótimas encontradas, de um total de 810 instâncias, o Modelo 1 encontrou 411 soluções ótimas, o $PS1$, $PS2$ e $PS2_{RINS}$ encontram, respectivamente, 401, 423 e 265. A Tabela 7.8 compara o número de soluções ótimas encontradas utilizando a heurística construtiva HC7 e também o IG como solução inicial. Utilizando o IG, o número de soluções ótimas encontradas foi um pouco maior do que utilizando a heurística HC7.

De acordo com os experimentos apresentados, os métodos desenvolvidos uti-

Tabela 7.7. Tempo de execução médio para o Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$ com HC7 e IG como solução inicial.

		$HC7 + método$ (Tempo (s))				$IG + método$ (Tempo (s))			
n	m	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$	Modelo 1	$PS1$	$PS2$	$PS2_{RINS}$
15	2	252,49	261,15	**149,98**	499,82	300,70	324,99	215,39	500,83
15	5	**1,35**	1,85	1,86	29,25	100,46	100,41	100,41	133,03
15	10	**0,19**	1,30	1,09	0,99	100,21	100,17	100,26	100,48
20	2	**477,77**	486,65	481,16	583,10	496,38	506,68	502,64	567,50
20	5	96,52	81,16	**40,59**	331,49	131,61	150,51	110,99	409,68
20	10	**1,00**	4,05	3,89	16,42	100,56	100,51	100,68	110,22
25	2	**534,29**	562,36	534,86	591,47	547,50	568,72	552,57	589,33
25	5	358,24	321,78	**291,01**	509,72	319,04	345,47	309,20	552,86
25	10	**10,98**	12,91	11,90	134,22	103,64	102,82	103,62	212,67
30	2	**590,10**	600,00	600,00	600,00	591,57	600,00	599,07	600,00
30	5	552,36	537,39	**526,95**	580,44	529,60	550,31	533,94	593,87
30	10	149,64	87,98	**79,59**	357,97	140,47	149,55	148,51	396,52
35	2	**599,85**	600,00	600,00	600,00	600,00	600,00	600,00	600,00
35	5	565,74	563,59	**548,27**	586,81	562,49	549,24	555,22	588,97
35	10	378,11	287,63	**274,34**	494,26	309,90	303,13	288,89	517,35
40	2	**600,00**	**600,00**	**600,00**	**600,00**	**600,00**	**600,00**	**600,00**	**600,00**
40	5	589,35	579,21	**576,60**	600,00	578,91	578,45	578,79	600,06
40	10	486,20	441,45	433,78	534,00	462,43	**426,48**	433,11	518,73
Médias		346,90	335,03	**319,81**	425,00	365,44	369,91	357,46	455,13

Tabela 7.8. Número de soluções ótimas encontradas para o Modelo 1, $PS1$, $PS2$ e $PS2_{RINS}$ utizando as heurísticas HC7 e IG como solução inicial.

	Solução Inicial	
Método	**HC7**	**IG(100s)**
$Modelo$ 1	373	**411**
$PS1$	386	**401**
$PS2$	418	**423**
$PS2_{RINS}$	**265**	**265**

lizando o IG como solução inicial apresentaram os menores valores de RPD. Considerando que o IG possui a heurística $HC7$ como solução inicial, o valor do RPI do IG em relação a heurística $HC7$ é igual a 21,65%. Com isso, o IG consegue fornecer soluções de maior qualidade. Além disso, os métodos apresentaram pequenas porcentagens de melhoramento (RPI) quando utilizaram o IG, ou seja, melhoraram muito pouco as soluções iniciais fornecidas. Esse resultado pode indicar que o espaço de busca foi muito reduzido devido a alta qualidade da solução inicial, gerando assim poucas ou nenhuma melhora. O que se esperava do PS utilizando uma solução inicial de alta qualidade fornecida pelo IG era conseguir encontrar um número maior de soluções ótimas dentro do tempo estipulado. Esse número foi maior se comparado ao PS utilizando a heurística $HC7$, porém, não foi uma diferença significativa. Além disso, não houve uma melhora no tempo médio de execução considerando os 100 segundos de execução do IG mais o limite de 500 segundos para o PS e o modelo.

7.7 Conclusão

Neste capítulo foram abordados dois algoritmos baseados nas meta-heurísticas GRASP e IG. Primeiramente, foi apresentado um método responsável por calcular o tempo máximo de conclusão das tarefas e também como tratar soluções inviáveis causadas pelas restrições de precedência. Foram desenvolvidas duas buscas locais, denotadas por $BL1$ e $BL2$, baseadas em movimentos de inserção (vizinhança de inserção), que foram utilizadas, respectivamente, pelo IG e pelo GRASP. A fase de construção do GRASP é baseada na heurística construtiva $HC7$, porém, as tarefas são selecionadas aleatoriamente de uma lista restrita de tarefas candidatas. O tamanho da lista é definida pelo parâmetro α que representa a porcentagem de tarefas melhor avaliadas de acordo com o menor tempo de conclusão. No IG, a etapa de *destruição-construção*, possui o parâmetro d_{max} que define o tamanho da destruição realizada na solução corrente. Também é utilizado o parâmetro *limit* para restringir o número de inserções sem melhora realizadas em cada máquina durante a fase de busca local. Todos os parâmetros utilizados foram calibrados através de testes computacionais e os resultados foram analisados por meio de testes estatísticos.

Foram realizados experimentos em instâncias de médio e grande porte. Em todos os experimentos, o IG conseguiu ter um desempenho significativamente melhor que o GRASP. Também foram feitos testes para comparar o desempenho do PS e do IG. Em um primeiro experimento, todas as versões implementadas para o PS, o Modelo 1 e o IG utilizaram a heurística construtiva HC7 como solução inicial. De acordo com o experimento, o IG conseguiu ter desempenho significativamente melhor, tanto na qualidade da solução final quanto na velocidade com que as soluções são melhoradas. Em outro experimento, as versões do PS e o Modelo 1 foram executados utilizando como solução inicial a melhor solução obtida após 100 segundos de execução do IG. O PS e o Modelo 1 apresentaram dificuldades em melhorar a solução inicial e também em provar a otimalidade das soluções.

Capítulo 8

Conclusões e Trabalhos Futuros

8.1 Conclusões

Neste trabalho foi abordado o problema de sequenciamento em máquinas paralelas não-relacionadas considerando restrição de precedência entre as tarefas e tempos de preparação dependentes da máquina e da sequência, representado pela notação: $R_m \mid prec, S_{ijk} \mid C_{max}$. O objetivo do problema é encontrar uma sequência de tarefas para cada máquina a fim de minimizar o tempo máximo de conclusão, conhecido como *makespan*.

O problema abordado possui grande importância por fazer parte do cenário de diferentes indústrias de manufatura, escritórios e organizações que gerenciam projetos. Além disso, o problema é importante pois pertence à classe de problemas NP-difícil e é um problema onde as características consideradas ainda não foram abordadas na literatura.

Para resolver este problema foram desenvolvidos três diferentes modelos de programação linear inteira mista (PLIM). O primeira modelo, denominado Modelo 1, é adaptado do modelo proposto por Rabadi et al. (2006) originalmente utilizado para o problema $R_m \mid S_{ijk} \mid C_{max}$. O segundo modelo, denominado Modelo 2 é uma adaptação do modelo proposto por Vallada & Ruiz (2011) para o problema $R_m \mid S_{ijk} \mid C_{max}$. Por fim, o modelo denominado Modelo 3 é adaptado de Liu (2013) originalmente utilizado para o problema $R_m \mid Prec \mid T$ com o objetivo de minimizar o tempo total de atraso. De acordo com os experimentos computacionais realizados, o Modelo 1 obteve o melhor desempenho em relação a qualidade da soluções, tempo médio de execução e número de soluções ótimas encontradas. Se for levado em consideração a complexidade das restrições, o Modelo 1 e o Modelo 2 são os modelos mais simples devido a sua estrutura e o menor número de restrições.

Já o Modelo 3, que utiliza outra estratégia de modelagem, necessitou de linearização e variáveis mais complexas, o que pode ter afetado o seu desempenho.

Com o objetivo de obter soluções de boa qualidade e em pouco tempo computacional, neste trabalho foi proposta uma heurística construtiva onde foram desenvolvidas sete regras de prioridades diferentes, resultando nas heurísticas denominadas de $HC1$ a $HC7$. Segundo os experimentos realizados, as heurísticas de $HC1$ a $HC7$ apresentaram diferenças significativas em relação a qualidade das soluções geradas, sendo que a heurística $HC7$ obteve o melhor desempenho. Além disso, a heurística $HC7$ obteve resultados significativamente melhores que o método que gera soluções aleatórias. As grandes diferenças na qualidade da solução ao empregar diferentes regras de prioridade evidencia a importância de uma regra de prioridade bem definida. Portanto, vale destacar que outras estratégias podem ser abordadas analisando diferentes características do problema e obter diferentes resultados.

A técnica de resolução de problemas de PLIM denominada *Proximity Search* (PS) foi empregada neste trabalho a fim de obter um número maior de soluções ótimas do problema. Foram desenvolvidas três versões do PS, denominadas de $PS1$, $PS2$, $PS2_{RINS}$. Nos experimentos computacionais realizados, o PS utiliza o Modelo 1 como base para a obtenção de uma solução melhorada a cada iteração do algoritmo. Para um conjunto de instâncias de médio porte, foi comparado o desempenho das três versões do PS e também do Modelo 1, tendo como solução inicial a heurística construtiva $HC7$. De acordo com os experimentos, o $PS2_{RINS}$ obteve o melhor percentual de melhoramento em relação a solução inicial fornecida pela heurística construtiva $HC7$, superando o Modelo 1 e as demais versões do PS. Também foi analisado o desempenho dos métodos em relação ao tempo gasto para obter cada melhoria. Segundo os resultados, o $PS2_{RINS}$ consegue obter as melhores soluções em poucos segundos de execução. No entanto, comparando o tempo médio para finalizar a execução de cada instância do problema, o $PS2$ conseguiu obter a menor média do tempo total de execução e encontrou o maior número de soluções ótimas. Dentre os métodos exatos desenvolvidos, o PS apresentou-se como uma boa opção para obter soluções melhoradas gastando menor tempo computacional. O ponto de destaque desse método é a função de proximidade e a estratégia de corte aplicada, que possibilita resolver o modelo implementado internamente de forma mais rápida. Mas vale destacar que o desempenho do método pode ser afetado aplicando uma nova função de proximidade e definindo uma estratégia de atualização do tamanho do corte realizado.

Neste trabalho também foram desenvolvidos dois algoritmos heurísticos baseados nas meta-heurísticas GRASP e IG. Também foram desenvolvidas duas buscas

locais, denotadas por BL1 e BL2, utilizadas respectivamente pelo GRASP e IG. Todos os parâmetros utilizados foram calibrados e analisados estatisticamente. De acordo com os experimentos realizados, a qualidade das soluções encontradas pelo IG é superior as encontradas pelo GRASP. Experimentos realizados também mostraram que o IG apresentou melhores resultados que o PS. O IG encontra melhores soluções em poucos instantes de tempo de execução. Em um último experimento, o Modelo 1 e as versões do PS foram executadas utilizando como solução inicial a melhor soluções encontrada pelo IG após 100 segundos de execução. De acordo com o experimento, o Modelo 1 e as versões do PS melhoraram pouco a solução inicial e conseguiram provar otimalidade em aproximadamente metade das instâncias utilizadas. Isso mostra que o IG está fornecendo soluções de alta qualidade e que também podem ser soluções ótimas, diminuindo o espaço de busca dos dois métodos.

8.2 Trabalhos Futuros

Como trabalhos futuros são sugeridos os seguintes:

- Realizar novos testes e calibrar o valor do parâmetro θ empregado nas versões implementadas do *Proximity Search*;
- Realizar novos testes e calibrar a frequência com que é aplicada a heurística RINS na versão $PS2_{RINS}$ do *Proximity Search*;
- Propor um novo modelo matemático mais eficiente e aplicar novas técnicas de resolução exatas a fim de melhorar o desempenho do *Proximity Search*;
- Desenvolver novos algoritmos baseados na meta-heurística *Iterated Greedy* (IG) e empregar heurísticas híbridas a fim de obter um melhor desempenho;
- Implementar novas buscas locais testando outras estruturas de vizinhança, buscando melhores estratégias para tratar movimentos que resultam em soluções inviáveis e também criar meios para evitar movimentos não promissores.

Referências Bibliográficas

Achterberg, T.; Berthold, T. & Hendel, G. (2012). Rounding and propagation heuristics for mixed integer programming. Em *Operations Research Proceedings 2011*, pp. 71--76. Springer.

Arroyo, J. & Armentano, V. (2004). A partial enumeration heuristic for multi-objective flowshop scheduling problems. *Journal of the Operational Research Society*, 55(9):1000--1007.

Berthold, T. (2013). Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611--614.

Brucker, P.; Hurink, J. & Kubiak, W. (1999). Scheduling identical jobs with chain precedence constraints on two uniform machines. *Mathematical Methods of Operations Research*, 49(2):211--219.

Danna, E.; Rothberg, E. & Le Pape, C. (2005). Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71--90.

Driessel, R. & Mönch, L. (2011). Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. *Computers & Industrial Engineering*, 61(2):336--345.

Feo, T. A. & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109--133.

Fischetti, M. & Monaci, M. (2014). Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709--731.

Gacias, B.; Artigues, C. & Lopez, P. (2010). Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, 37(12):2141--2151.

Gary, M. R. & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of np-completeness. 1979. *WH Freman and Co*.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K. & Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287--326.

Hassan Abdel-Jabbar, M.-A.; Kacem, I. & Martin, S. (2014). Unrelated parallel machines with precedence constraints: application to cloud computing. Em *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pp. 438--442. IEEE.

Herrmann, J.; Proth, J.-M. & Sauer, N. (1997). Heuristics for unrelated machine scheduling with precedence constraints. *European Journal of Operational Research*, 102(3):528--537.

Huo, Y. & Leung, J. Y. (2005). Online scheduling of precedence constrained tasks. *SIAM Journal on Computing*, 34(3):743--762.

Kim, E.-S. (2011). Scheduling of uniform parallel machines with s-precedence constraints. *Mathematical and Computer Modelling*, 54(1):576--583.

Kim, E.-S.; Sung, C.-S. & Lee, I.-S. (2009). Scheduling of parallel machines to minimize total completion time subject to s-precedence constraints. *Computers & Operations Research*, 36(3):698--710.

Kumar, V. A.; Marathe, M. V.; Parthasarathy, S. & Srinivasan, A. (2009). Scheduling on unrelated machines under tree-like precedence constraints. *Algorithmica*, 55(1):205--226.

Lin, Y.; Pfund, M. E. & Fowler, J. W. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers & Operations Research*, 38(6):901--916.

Liu, C. (2013). A hybrid genetic algorithm to minimize total tardiness for unrelated parallel machine scheduling with precedence constraints. *Mathematical Problems in Engineering*, 2013.

Liu, C. & Yang, S. (2011). A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *Journal of Software*, 6(6):1146--1153.

Manne, A. S. (1960). On the job-shop scheduling problem. *Operations Research*, 8(2):219--223.

Miller Jr, R. G. (1997). *Beyond ANOVA: basics of applied statistics*. CRC Press.

Nouri, M. & Ghodsi, M. (2012). Scheduling tasks with exponential duration on unrelated parallel machines. *Discrete Applied Mathematics*, 160(16):2462--2473.

Osman, I. & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551--557.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

Queyranne, M. & Schulz, A. S. (2006). Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM Journal on Computing*, 35(5):1241--1253.

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

Ramachandra, G. & Elmaghraby, S. E. (2006). Sequencing precedence-related jobs on two machines to minimize the weighted completion time. *International Journal of Production Economics*, 100(1):44--58.

Ruiz, R. & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033--2049.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

Vallada, 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.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131--140.

Weng, M. X.; Lu, J. & Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International journal of production economics*, 70(3):215--226.

Woeginger, G. J. (2000). A comment on scheduling on uniform machines under chain-type precedence constraints. *Operations Research Letters*, 26(3):107--109.

Yuan, J.; Li, W. & Yuan, J. (2012). A best possible online algorithm for scheduling equal-length jobs on two machines with chain precedence constraints. *Theoretical Computer Science*, 457:174--180.

Apêndice A

Detalhes do desenvolvimento dos métodos e experimentos

A.1 Implementação da Função de Proximidade

Nesta seção será demonstrado como implementar a função de proximidade, definida no Capítulo 6, utilizando recursos da biblioteca IBM Concert.

Inicialmente, considere duas solução X e X' definidas, respectivamente, pelas variáveis x e x' que representam uma sequência de $j \in J$ valores binários (0-1). A seguir é apresentada a equação para obter a função de proximidade entre as duas soluções:

$$\Delta(X, X') = \sum_{j \in J : x'_j = 0} x_j + \sum_{j \in J : x'_j = 1} (1 - x_j) \tag{A.1}$$

Como exemplo, considere o conteúdo de $x'_j = \{1, 0, 1, 0, 1, 1\}$. Aplicando o valor de x' sobre a equação, o resultado é apresentado a seguir:

$$\Delta(X, X') = (1 - x_1) + x_2 + (1 - x_3) + x_4 + (1 - x_5) + (1 - x_6) \tag{A.2}$$

$$\Delta(X, X') = 1 - x_1 + x_2 + 1 - x_3 + x_4 + 1 - x_5 + 1 - x_6 \tag{A.3}$$

$$\Delta(X, X') = -x_1 + x_2 - x_3 + x_4 - x_5 - x_6 + 4 \tag{A.4}$$

Como pode ser observado no resultado da equação, se o valor de x'_j é igual a 0, o coeficiente de x_j fica positivo, caso contrário, o coeficiente de x_j fica negativo. Ao final, é somado o valor 4, que corresponde a quantidade de valores iguais a 1 em x'.

Baseado nesta característica, foi criada uma estratégia para definir a função de proximidade do PS para o problema abordado, no entanto, considerando as variáveis x'_{ijk} e x_{ijk}. O valor de x'_{ijk} é fornecido pela solução corrente do PS e o valor de x_{ijk} será encontrado pelo modelo matemático implementado internamente pelo PS. A equação da função de proximidade do problema abordado é definida da seguinte forma:

$$\Delta(X, X') = \sum_{i \in M} \sum_{j \in J_0} \sum_{k \in J : x'_{ijk} = 0} x_{ijk} + \sum_{i \in M} \sum_{j \in J_0} \sum_{k \in J : x'_{ijk} = 1} (1 - x_{ijk}) \qquad \text{(A.5)}$$

A estratégia para formação da função de proximidade é alterar o coeficiente da variável x_{ijk} de acordo com o valor de x'_{ijk}: se $x'_{ijk} = 0$, o coeficiente de x_{ijk} é alterado para 1, caso contrário, o coeficiente de x_{ijk} é alterado para -1. Ao final, é acrescido à função de proximidade o valor da variável n que representa o número total de tarefas e que também corresponde ao número de valores iguais a 1 contidos na variável x'_{ijk}. Para o $PS2$ e $PS2_{RINS}$ também é necessário somar a função de proximidade à multiplicação entre a variável Z e M. A seguir é apresentado o trecho de código para implementação da função de proximidade do $PS2$ ou $PS2_{RINS}$ utilizando a linguagem C++ e a biblioteca IBM Concert (considere $x1 = x'$).

```
// Cria uma função objetivo de minimização para o modelo.
IloObjective funcaoProximidade = IloAdd(model, IloMinimize(env));
for (int i = 1; i <= m; i++){
     for (int j = 0; j <= n; j++){
          for (int k = 1; k <= n; k++){
               if (x1[i][j][k] == 0){
                    // altera o coeficiente de x_ijk para 1.
                    funcaoProximidade.setLinearCoef(x[i][j][k], 1);
               }
               else if (x1[i][j][k] == 1){
                    // altera o coeficiente de x_ijk para -1.
                    funcaoProximidade.setLinearCoef(x[i][j][k], -1);
               }
          }
     }
}
// adiciona um valor constante n a função objetivo.
funcaoProximidade.setConstant(n);
// adiciona o valor de Z * M na função objetivo.
funcaoProximidade.setLinearCoef(Z, M);
```

Figura A.1. Código-fonte da função de proximidade.

Para atualizar a função de proximidade devido a atualização do valor de x'_{ijk} provocada pelas iterações do PS, basta apenas redefinir o conteúdo dos coeficientes de x_{ijk} em cada iteração do PS.

A.2 Definição dos Parâmetros

Para implementação e execução do modelo matemático e do PS foram necessários definir alguns parâmetros específicos do CPLEX que são apresentados através da Tabela A.1. Os parâmetros são definidos na linguagem C++ de acordo com a biblioteca IBM Concert.

Tabela A.1. Lista de parâmetros utilizados pelos modelos matemáticos e versões do PS

cplex.setParam(IloCplex::ClockType, 2);	Parâmetro igual a 2 permite cronometrar o tempo real de execução (wall-clock time).
cplex.setParam(IloCplex::TiLim, 600);	Permite definir o tempo máximo de execução em segundos do modelo matemático. Ex.: 600 segundos.
cplex.setParam(IloCplex::IntSolLim, 1);	Permite finalizar a execução do modelo após encontrar um determinado número de soluções incumbentes. No $PS1$ este parâmetro é definido igual a 1, já no $PS2$ e $PS2_{RINS}$ é definido igual a 2, pois a primeira solução incumbente é igual a solução corrente.
cplex.setParam(IloCplex::RINSHeur, 1);	Este parâmetro habilita a heurística RINS implementada internamente pelo CPLEX e utilizada pelo $PS2_{RINS}$. O parâmetro igual a 1, define que a heurística será aplicada a cada nó da árvore de busca.
IloInt M = n x 2;	A variável M é utilizada para penalizar a variável z utilizada na função objetivo do PS2 e $PS2_{RINS}$. Este valor assume n (número de tarefas) x 2, pois é o valor máximo da função de proximidade entre duas soluções do problema abordado.
cplex.setParam(IloCplex::EpAGap, M / 2);	Este parâmetro foi definido para M/2 a fim de abortar a execução do modelo no $PS2$ e $PS2_{RINS}$ assim que uma solução viável com z = 0 for encontrada.

A.3 Cálculo da métrica Primal Integral utilizada pelos experimentos computacionais

Para realizar os experimentos analisando a métrica *Primal Integral* é necessário obter dois dados a cada melhoria da solução corrente de um método de resolução. O primeiro dado é o valor da função objetivo da solução melhorada e o segundo é o tempo gasto para obter essa solução.

Na Tabela A.2 é apresentado, como exemplo, o arquivo de saída do método $PS2_{RINS}$ contendo a função objetivo e o tempo gasto de cada melhoria encontrada para uma dada instância do problema.

Tabela A.2. Função objetivo e tempo gasto em cada solução de melhora obtida pelo $PS2_{RINS}$

Função Objetivo (C_{max})	**Tempo (s)**
373	0,00
368	4,33
366	4,84
363	5,38
359	8,91
354	9,50
345	11,41
329	13,17
326	29,96
318	63,96
286	121,14
282	268,45
282	600,00

Para obter a função objetivo e o tempo gasto a cada solução de melhoria (incumbente) encontrada pelos modelos matemáticos implementados neste trabalho, foi preciso definir os seguintes parâmetros na implementação: *cplex.setParam(IloCplex::MIPDisplay, 3)*, utilizado para exibir no log do CPLEX a solução incumbente e o tempo gasto para encontrá-la, e *cplex.setOut(NomeArquivo)*, utilizado para exportar o log do CPLEX para um arquivo de texto.

Já no PS, o valor da função objetivo e o tempo gasto pode ser obtido em cada iteração do PS logo após a execução do modelo matemático implementado internamente pelo método. Isso deve-se ao fato que em cada resolução do modelo, o PS vai encontrar um solução melhorada, caso contrário, o PS é finalizado. Nas meta-heurísticas GRASP e IG, o dados podem ser obtidos no momento onde a melhor solução (S_{best}) é atualizada.

Considerando os dados apresentados pela Tabela A.2 como exemplo, o cálculo da métrica *Primal Integral* é definida e exemplificada a seguir. As equações da métrica *Primal Integral* (discutidas na seção 6.3.1) são:

$$p(t) = \frac{f(t) - f_{best}}{f_{best}} \tag{A.6}$$

$$P(t_{max}) = \int_0^{t_{max}} p(t) \times \Delta t \tag{A.7}$$

Para calcular o *Primal Integral* P(t) em um tempo t, considere as seguintes variáveis:

- b = número de melhoras obtidas;
- i = um número incremental que identifica a solução melhorada na ordem em que foi obtida pelo método aplicado;
- f_i = função objetivo da solução melhorada em i;
- f_{best} = função objetivo da melhor solução obtida para a instância;
- t_i = tempo gasto para obter a solução em i;
- $p(t_i)$ = função *gap* para no tempo t_i calculada a partir da equação A.6;
- $(t_{i+1} - t_i)$ = diferença entre o tempo gasto em t_{i+1} e t_i;
- $P(t_i)$ = valor da *Primal Integral* para o tempo t_i da solução em i;
- t_d = tempo desejado, ou seja, um tempo específico onde se deseja obter o valor da *Primal Integral*;
- $P(t_d)$ = valor do *Primal Integral* para o tempo desejado t_d;

A métrica Primal Integral $P(t)$ pode ser calculada pela seguinte equação:

$$\sum_{i=0}^{b-1} p(t_i) \times (t_{i+1} - t_i) \tag{A.8}$$

A Tabela A.3 apresenta o resultado do cálculo da *Primal Integral* para cada solução encontrada conforme equação A.8.Também é apresentado o valor da *Primal Integral* nos tempos desejados $t_d = \{5s, 10s, 30s, 60s, 120s, 300s, 600s\}$.

Para exemplificar o cálculo de P para um tempo desejado t_d, considere a Tabela A.3 e $t_d = 5\ s$. Primeiramente, deve-se obter o maior tempo t menor que 5, neste caso é $t_2 = 4{,}84\ s$. Para obter o valor de P(5) deve-se realizar o seguinte cálculo:

$$P(5) = P(t_1) + p(t_2) \times (5 - t_2) \tag{A.9}$$
$$P(5) = 1{,}553 + 0{,}298 \times (5 - 4{,}84) \tag{A.10}$$
$$P(5) = 1{,}553 + 0{,}298 \times 0{,}16 \tag{A.11}$$
$$P(5) = 1{,}553 + 0{,}047 \tag{A.12}$$
$$P(5) = 1{,}6 \tag{A.13}$$

Como pode ser observado, o valor de P(5) para o tempo $t_d = 5s$ é 1,6. A mesma estratégia deve ser aplicada para qualquer tempo na qual deseja-se conhecer o valor de P.

Tabela A.3. Demonstrativo do cálculo da *Primal Integral* para as soluções de melhoria obtidas pelo método $PS2_{RINS}$ para uma dada instância do problema.

i	f_i	t_i	$p(t_i)$	$\Delta(t_{i+1} - t_i)$	P(t_i)	t_d	P(t_d)
0	373	0	0,323	4,33	1,397		
1	368	4,33	0,305	0,51	1,553		
2	366	4,84	0,298	0,54	1,714	5	1,600
3	363	5,38	0,287	3,53	2,728		
4	359	8,91	0,273	0,59	2,889		
5	354	9,5	0,255	1,91	3,376	10	3,016
6	345	11,41	0,223	1,76	3,770		
7	329	13,17	0,167	16,79	6,568		
8	326	29,96	0,156	29,00	11,093	30	6,574
9	318	58,96	0,128	60,18	18,775	60	11,225
10	286	119,14	0,014	149,31	20,893	120	18,787
11	282	268,45	0,000	331,55	20,893	300	20,893
12	282	600	0,000	-	-	600	20,893

Apêndice B

Publicações

A seguir estão relacionados os trabalhos publicados em anais de eventos científicos realizados a partir desta dissertação:

1. **Título**: Heuristic Algorithms for Unrelated Parallel Machine Scheduling with Precedence Constraints and Sequence-Dependent Setup Times.
   Autores: Felippe M. Faêda, José Elias C. Arroyo, André Gustavo dos Santos, Michele Monaci.
   Evento: 11th Metaheuristics International Conference (MIC 2015).
   Local: Agadir, Marrocos.
   Período: 7 a 10 de junho de 2015.

2. **Título**: Sequenciamento de Tarefas em Máquinas Paralelas com Tempos de Preparação e Precedência entre as Tarefas: Modelagem e Heurísticas Construtivas. **Autores**: Felippe M. Faêda, José Elias C. Arroyo, André Gustavo dos Santos, Michele Monaci.
   Evento: XLVII Simpósio Brasileiro de Pesquisa Operacional (SBPO 2015).
   Local: Porto de Galinhas, Brasil.
   Período: 25 a 28 de agosto de 2015.