Programa de Pós-Graduação em Ciência da Computação
Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Pró-Reitoria de Pesquisa e Pós-Graduação
Universidade Federal de Ouro Preto

ALGORITMOS HEURÍSTICOS HÍBRIDOS PARA O PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto, como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Aluno: Matheus Nohra Haddad

Orientador: Marcone Jamilson Freitas Souza

Co-Orientador: Haroldo Gambini Santos

OURO PRETO-MG
2012

ALGORITMOS HEURÍSTICOS HÍBRIDOS PARA O PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

Matheus Nohra Haddad

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto, como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Aprovada por:

Marcone Jamilson Freitas Souza, D.Sc. / DECOM-UFOP
(Orientador)

Haroldo Gambini Santos, D.Sc. / DECOM-UFOP
(Co-orientador)

José Elias Cláudio Arroyo, D.Sc. / DPI-UFV

Frederico Gadelha Guimarães, D.Sc. / DEE-UFMG

Alexandre Xavier Martins, D.Sc. / DEENP-UFOP

Ouro Preto, 15 de fevereiro de 2012.

À minha mãe Kika e meu pai Aloisio

Agradecimentos

Aos meus pais Aloisio e Kika pela brilhante educação passada e pelo incentivo.

Aos meus irmãos Thais e Eduardo pelo carinho e união.

Aos meus avós Rafick (em sua memória) e Maria, Ivo e Laila pelo amor transmitido e apoio.

À Cláudia pela grande contribuição na minha formação, tanto acadêmica quanto pessoal.

À minha namorada Fernanda por todo amor, amizade, incentivo e por ter aparecido em minha vida durante o mestrado.

À minha família pelo apoio e incentivo.

Aos grandes amigos formados em Ouro Preto, em especial Samuel, Igor, Thais, Pablo, Sabir, Akinase, Kurumin, Larissa, Peruboy, Marcelo e muitos outros que estavam sempre dispostos a ajudar, além de confiarem em mim.

Aos excepcionais professores que possibilitaram tal feito, em especial Fred, Alexandre, Haroldo por sanar minhas dúvidas e me ajudarem sempre.

Ao amigo e orientador Marcone pela confiança depositada em mim, pela responsabilidade e por ter sido um verdadeiro orientador, tentando sempre construir o melhor caminho para a elaboração da pesquisa científica.

Aos inseparáveis amigos de São João del Rei, Lucas, Rodrigo, Marcelo, Marcelloni, Rafael, Paulo, Bernardo, Atemir e muitos outros por estarem sempre a meu lado.

Aos companheiros do Beco da Mãe Chica, Bob e Breno, por terem me acolhido e por se tornarem meus grandes amigos.

A todas indignadas da República Indignação por terem me escolhido e me proporcionarem ótimos momentos.

Resumo

Este trabalho aborda o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência, do inglês *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*, ou apenas UPMSPST. Neste problema há um conjunto de tarefas e máquinas e a cada tarefa está associado um tempo de processamento, que é diferente para cada máquina. Dadas duas tarefas, também existe um tempo de preparação dependente da sequência delas e da máquina utilizada. O objetivo considerado neste problema é minimizar o tempo máximo de conclusão do sequenciamento, o chamado *makespan*. O UPMSPST é muito encontrado no âmbito industrial e pertence à classe NP-difícil. Visando a sua resolução, são propostos três algoritmos heurísticos híbridos. O primeiro, nomeado IVP, combina os procedimentos heurísticos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Path Relinking* (PR). O segundo, nomeado GIVP, difere do primeiro pelo fato de gerar a solução inicial pela fase de construção *Greedy Randomized Adaptive Search Procedure* (GRASP), e não por um procedimento guloso. O terceiro algoritmo, nomeado GIVMP, difere dos outros em três estratégias: a fase PR utiliza a estratégia *mixed relinking* ao invés de *backward relinking*, as vizinhanças que formam o VND são usadas em ordem aleatória a cada chamada e não em ordem previamente definida; por fim, o GIVMP inclui um módulo de busca local feita pelo resolvedor CPLEX 12.1 de programação matemática. Este resolvedor é aplicado apenas à máquina gargalo do problema e utiliza um modelo de programação inteira mista baseado no problema do Caixeiro Viajante Assimétrico. Para explorar o espaço de soluções são utilizados movimentos de inserção e troca de tarefas entre máquinas. Para testar os algoritmos foram utilizados dois conjuntos de problemas-teste da literatura. Inicialmente eles foram comparados entre si em um conjunto reduzido de instâncias, tendo-se verificado a superioridade do algoritmo GIVMP. Em seguida, este algoritmo foi comparado com outros seis da literatura, sendo dois no primeiro conjunto de problemas-teste e outros quatro no segundo. No primeiro conjunto verificou-se a superioridade absoluta do GIVMP sobre os dois algoritmos genéticos da literatura, tendo-se encontrado desvios percentuais relativos médios menores e novas melhores soluções. No segundo conjunto de problemas-teste verificou-se que o GIVMP tem desempenho inferior a um dos algoritmos, mas superior em relação aos outros três. Observa-se, no entanto, que neste segundo conjunto de problemas, não foram disponibilizados os desvios percentuais relativos médios dos algoritmos, mas apenas os desvios dos melhores valores encontrados, impedindo uma comparação de robustez dos algoritmos.

PALAVRAS-CHAVE: Sequenciamento em máquinas paralelas, *makespan, Iterated Local Search, Path Relinking*, VND, GRASP, tempos de preparação

Abstract

This work addresses the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times, or just UPMSPST. In this problem there is a set of jobs and machines and for each job there is a processing time associated, which is different for each machine. Given two jobs, there is also a setup time that depends on their sequence and on the machine used. The objective considered in this problem is to minimize the maximum completion time of the schedule, the so-called *makespan*. The UPMSPST is often found in industries and belongs to the NP-hard class. Aiming to its resolution, three hybrid heuristic algorithms are proposed. The first, named IVP, combines the heuristic procedures *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) and *Path relinking* (PR). The second, named GIVP, differs from the first because it generates the initial solution by the construction phase *Greedy Randomized Adaptive Search Procedure* (GRASP), and not by a greedy procedure. The third algorithm, named GIVMP, differs in three strategies: the phase PR uses the strategy *mixed relinking* instead of *backward relinking*, the neighborhoods that form the VND are used in random order at each call and not in previously defined order; finally, the GIVMP includes a local search module made by CPLEX 12.1 mathematical programming solver. This solver is applied only to the bottleneck machine of the problem and uses a mixed integer programming model based on the Asymmetric Travelling Salesman problem. To explore the solution space, insertion and swap movements between machines are used. In order to test the algorithms, two sets of test problems from the literature are used. Initially they were compared with each other on a reduced set of instances, and it was found the superiority of the GIVMP algorithm. Then, this algorithm was compared with six other in literature, being two in the first set of test problems and four others in the second. In the first set it was found the superiority of GIVMP on two genetic algorithms of the literature, having been found lower relative percentage deviations and new best solutions. In the second set of test problems it was showed that the performance of GIVMP is lower than one of the algorithms, but higher than the other three. It is observed, however, that in this second set of problems, the average relative percentage deviations of the algorithms were not available, but only the deviations of the best values found, which impedes a comparison of robustness of the algorithms.

Keywords: unrelated parallel machine scheduling, makespan, Iterated Local Search, Path Relinking, VND, GRASP, setup times

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACO *Ant Colony Optimization* implementado em Arnaout et al. (2010)

ANOVA *Analysis of Variance* - Análise de Variância

ASPT *Adaptive Shortest Processing Time*

GA1 Algoritmo Genético implementado em Vallada e Ruiz (2011)

GA2 Algoritmo Genético implementado em Vallada e Ruiz (2011)

GADP Algoritmo Genético implementado em Chang e Chen (2011)

GIVMP Algoritmo baseado no GRASP, ILS, VND, PIM e PR

GIVP Algoritmo baseado no GRASP, ILS, VND e PR

GRASP *Greedy Randomized Adaptive Search Procedure*

ILS *Iterated Local Search*

IVP Algoritmo baseado no ILS, VND e PR

LC Lista de Tarefas Candidatas

LB *Lower Bound* - Limite Inferior

LRC Lista Restrita de Candidatos

Meta-RaPS *Meta-heuristic for Randomized Priority Search* implementado em Rabadi et al. (2006)

PCVA Problema do Caixeiro Viajante Assimétrico

PH *Partitioning Heuristic* implementado em Helal et al. (2006)

PIM *Mixed Integer Programming* - Programação Inteira Mista

PMSP Problema de Sequenciamento em Máquinas Paralelas

PR *Path Relinking*

PSM Problema de encontrar a melhor sequência das tarefas em uma máquina

RPD *Relative Percentage Deviation* - Desvio Percentual Relativo

RPD_{avg} Desvio Percentual Relativo Médio

RPD_{best} Desvio Percentual Relativo à Melhor Solução

RSA *Restricted Simulated Annealing* implementado em Ying et al. (2010)

RVND *Random Variable Neighborhood Descent*

SADP *Simulated Annealing* implementado em Chang e Chen (2011)

SAP *Shortest Adjusted Processing Time* implementado em Rabadi et al. (2006)

SL *Smallest Load* implementado em Rabadi et al. (2006)

SSB1 Formulação de Sarin et al. (2005)

UPMSPST *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times* - Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Depentedes da Sequência

VND *Variable Neighborhood Descent*

VNS *Variable Neighborhood Search*

Capítulo 1

Introdução

A busca pela eficiência em processos industriais está se intensificando cada vez mais. Tal fato se deve à grande competitividade promovida através da globalização. A fim de se ter eficiência é essencial ter um bom planejamento da produção. Aumento da produtividade e redução de custos são consequências de um bom planejamento. Ferramentas computacionais que auxiliam o desenvolvimento de um planejamento estão cada vez mais sendo utilizadas por indústrias.

Planejar o sequenciamento de tarefas em várias máquinas é um problema constantemente encontrado no âmbito industrial. O problema consiste em definir uma sequência de tarefas que devem ser alocadas em máquinas paralelas de forma que o tempo máximo de conclusão do processo seja o menor possível. Tais máquinas podem ser consideradas idênticas, uniformes e não-relacionadas. São ditas idênticas quando o tempo de processamento para a mesma tarefa em outra máquina é idêntico, uniformes quando os tempos de processamento das tarefas nas máquinas variam com uma certa proporcionalidade e não-relacionadas quando o tempo de processamento para a mesma tarefa em outra máquina é diferente.

Dentre os vários problemas de sequenciamento em máquinas paralelas, destaca-se o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (em inglês *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*), ou apenas UPMSPST. Este problema se caracteriza por conter tempos de preparação entre as tarefas que são dependentes da máquina a qual as mesmas estão alocadas. Também existem tempos de preparação que são dependentes da sequência em que essas tarefas estão alocadas nas máquinas. O objetivo é minimizar o tempo máximo de conclusão do sequenciamento, o chamado *makespan*.

Considera-se que o UPMSPST tem importância tanto teórica quanto prática. Teórica, pois é um problema de difícil solução por pertencer à classe NP-difícil e prática, pois existem muitas situações onde ele aparece, como por exemplo, em processos de fabricações em indústrias têxteis (Pereira Lopes e de Carvalho, 2007).

Devido a dificuldade de encontrar soluções ótimas para o UPMSPST em tempos computacionais restritos, a utilização de heurísticas para se obter soluções de qualidade deve ser considerada. Na literatura são encontradas várias abordagens heurísticas que procuram resolver esse problema e problemas semelhantes. Estas abordagens serviram de inspiração para o desenvolvimento do presente trabalho.

Rabadi et al. (2006) e Vallada e Ruiz (2011) estão entre os principais trabalhos

que abordam o UPMSPST. Em Rabadi et al. (2006), é realizada a implementação de uma metaheurística para busca randômica priorizada. Esta metaheurística utiliza uma estratégia que combina heurísticas de construção e refinamento, utilizando ideias semelhantes ao *Greedy Randomized Adaptive Search Procedure* – GRASP (Feo e Resende, 1995). Em Vallada e Ruiz (2011), é proposta a resolução do UPMSPST por meio de Algoritmos Genéticos (Holland, 1975; Goldberg, 1989), cuja principal característica é realizar cruzamentos com buscas locais "limitadas". Em ambos os trabalhos são criados e disponibilizados problemas-teste para o UPMSPST, o primeiro os disponibiliza em SchedulingResearch (2005) e o segundo em SOA (2011).

Baseando-se nesses trabalhos e nos problemas-teste citados, o presente trabalho busca um algoritmo heurístico que seja capaz de resolver o UPMSPST de forma eficiente, ou seja, um algoritmo que possa obter soluções de melhor qualidade para o UPMSPST em um tempo considerável. A medida de qualidade de uma solução do UPMSPST, ou o critério de otimização considerado é o *makespan*.

Com esse intuito, são desenvolvidos três algoritmos heurísticos híbridos, isto é, que reúnem várias metaheurísticas e técnicas de intensificação. O objetivo da criação de um algoritmo heurístico híbrido descende da ideia de que um determinado procedimento de otimização, por si só, pode não ser muito eficiente para resolver um problema, porém ao combinar as características positivas de vários procedimentos de uma maneira inteligente pode-se constituir um algoritmo bastante eficiente.

O primeiro algoritmo proposto é nomeado IVP e tem como base o *Iterated Local Search* – ILS (Lourenço et al., 2003), porém com uma diferença para a maioria das aplicações de ILS, otimizando subpartes do problema a cada perturbação. A geração da solução inicial é feita a partir da heurística de construção gulosa *Adaptive Shortest Processing Time* – ASPT, que é uma adapatação da heurística *Shortest Processing Time* – SPT (Baker, 1974). Na intenção de tornar mais eficiente a exploração do espaço de soluções, utilizou-se o método *Variable Neighborhood Descent* – VND (Mladenovic e Hansen, 1997), responsável por realizar as buscas locais do ILS. Estas buscas locais são feitas com uma certa ordem de processamento, a partir de movimentos de trocas e inserções de tarefas. Além disso, também é incluído no algoritmo a aplicação da técnica *Path Relinking* – PR (Glover, 1996) a fim de intensificar e diversificar a busca no espaço de soluções, interligando caminhos entre soluções de qualidade. O PR é aplicado com a estratégia *backward relinking*, a qual constrói um caminho partindo de uma solução "melhor" até chegar a uma solução "pior".

O segundo algoritmo proposto, de nome GIVP, é uma modificação do primeiro (IVP) ao substituir a construção gulosa por uma construção parcialmente gulosa. Nesta construção utiliza-se a fase de construção do método GRASP. Ao fazer isto, visa-se a obter, ao mesmo tempo, as características positivas das soluções geradas tanto de forma gulosa quanto de forma aleatória. As soluções gulosas favorecem uma convergência rápida por estarem geralmente próximas a ótimos locais; já a solução gerada de forma aleatória possibilita, por conta de sua diversidade, uma maior exploração do espaço de busca.

O GIVMP é o terceiro algoritmo proposto e é uma modificação do segundo (GIVP) ao incorporar um modelo de Programação Inteira Mista (PIM). Este modelo PIM é aplicado ao final do ILS como estratégia de intensificação, buscando melhorar a melhor solução encontrada. O modelo PIM proposto tem por objetivo

otimizar a máquina gargalo, ou seja, a máquina com maior tempo de conclusão (*makespan*). Otimizar as tarefas de uma máquina no UPMSPST é equivalente a resolver o Problema do Caixeiro Viajante Assimétrico (PCVA). Por isso, o modelo PIM utilizado no GIVMP é o mesmo usado para solucionar o PCVA. Outra modificação do GIVMP é a utilização do VND sem definir inicialmente a ordem de processamento das buscas locais, sendo esta ordem definida aleatoriamente a cada chamada do método. Destaca-se ainda a substituição da estratégia *backward relinking* pela estratégia *mixed relinking*. Tal estratégia é caracterizada por explorar dois caminhos, simultaneamente, partindo de duas soluções. O fim dessa estratégia de PR se dá quando esses caminhos se encontram em uma solução equidistante de ambas soluções.

1.1 Objetivos

Nesta seção são apresentados os principais objetivos deste trabalho.

1.1.1 Objetivo Geral

O presente trabalho visa ao estudo do problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (UPMSPST). Tem-se como objetivo principal desenvolver um algoritmo heurístico híbrido robusto que seja capaz de encontrar soluções de boa qualidade para tal problema rapidamente, além de ser competitivo com os melhores algoritmos encontrados na literatura.

1.1.2 Objetivos Específicos

Para que se possa atingir o objetivo geral é necessário o cumprimento dos seguintes objetivos específicos:

- Analisar os trabalhos da literatura que tratam o problema abordado e problemas relacionados, bem como as estratégias utilizadas para resolvê-los;
- Desenvolver algoritmos heurísticos híbridos, baseado em metaheurísticas distintas, para resolver o problema de sequenciamento apresentado;
- Resolver os problemas-teste encontrados na literatura com o intuito de validar os resultados encontrados pelos algoritmos desenvolvidos, comparando-os com os resultados da literatura;
- Divulgar os resultados obtidos em eventos científicos, bem como, possivelmente, em periódicos da área.

1.2 Motivação

A grande concorrência existente no âmbito industrial faz com que cada vez mais as indústrias necessitem de ferramentas tecnológicas para auxiliar em tomadas de

decisões. Com o auxílio dessas ferramentas os processos produtivos se tornam mais eficientes, facilitando a gestão industrial. Métodos computacionais que procuram resolver problemas de sequenciamento em máquinas são de grande importância para indústrias.

Problemas de sequenciamento em várias máquinas, em especial o UPMSPST, são percebidos em indústrias têxteis, químicas, eletrônicas, entre outras. A aplicação prática instiga o estudo de soluções para esse problema.

No ramo da pesquisa, também se tem a motivação por conta de o UPMSPST ser um problema da classe NP-difícil. Tal fato induz pesquisas envolvendo soluções heurísticas na tentativa de solucioná-lo de forma eficiente.

1.3 Estrutura do Trabalho

O restante deste trabalho está estruturado como segue.

No Capítulo 2 o problema em questão é caracterizado. A revisão bibliográfica é realizada no Capítulo 3, destacando as principais fontes que serviram de inspiração para o desenvolvimento deste trabalho. O Capítulo 4 apresenta a metodologia utilizada para o desenvolvimento deste trabalho. Resultados computacionais são apresentados e discutidos no Capítulo 5. Finalmente, no Capítulo 6, conclui-se este trabalho, bem como são apresentadas possíveis propostas a serem exploradas em trabalhos futuros.

Capítulo 2

Caracterização do Problema

Neste Capítulo, primeiramente é definido o Problema de Sequenciamento em Máquinas Paralelas, PMSP (Seção 2.1). Em seguida, na Seção 2.2, são descritas as características do Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência (UPMSPST), que é abordado neste trabalho.

2.1 Problema de Sequenciamento em Máquinas Paralelas

O problema de sequenciamento em máquinas paralelas (PMSP, das iniciais em inglês de *Parallel Machine Scheduling Problem*) foi introduzido em McNaughton (1959) e desde então ele tem sido tema de estudos no âmbito da pesquisa científica.

Este problema é caracterizado por conter uma quantidade de tarefas que devem ser alocadas em uma quantidade pré-fixada de máquinas. O objetivo desse problema não só envolve saber em qual máquina cada tarefa será alocada, mas também deve ser definida uma sequência dessas tarefas nessas máquinas.

Os PMSPs são classificados em três categorias (Cheng e Sin, 1990): máquinas paralelas idênticas, máquinas paralelas uniformes e máquinas paralelas não-relacionadas. Nas máquinas paralelas idênticas, os tempos de processamento de cada tarefa são iguais para todas as máquinas. Já nas máquinas paralelas uniformes, os tempos de processamento das tarefas nas máquinas variam com uma certa proporcionalidade. Por fim, nas máquinas paralelas não-relacionadas, os tempos de processamento de cada tarefa em cada máquina são completamente diferentes, não possuindo nenhum padrão entre eles.

Outra classificação dos PMSPs se dá quanto ao critério de otimização, que pode ser, segundo Cheng e Sin (1990):

(a) Baseado em tempo de conclusão:

  (i) Tempo de conclusão total ou tempo médio de conclusão.

  (ii) Tempo de conclusão ponderado.

  (iii) Tempo máximo de conclusão, o chamado *makespan*.

(b) Baseado em data de entrega:

(i) *Lateness* total das tarefas ou *lateness* médio das tarefas.

(ii) *Lateness* máximo das tarefas.

(iii) Atraso total das tarefas ou atraso médio das tarefas.

(iv) Atraso ponderado.

(v) Atraso máximo das tarefas.

(c) Baseado em fluxo de tempo:

(i) Fluxo de tempo total das tarefas ou fluxo médio das tarefas.

(ii) Fluxo de tempo ponderado.

2.2 Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência

No problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (UPMSPST) tem-se um conjunto $N = \{1, ..., n\}$ de n tarefas e um conjunto $M = \{1, ..., m\}$ de m máquinas, com as seguintes características:

(a) Cada tarefa deve ser processada exatamente uma vez por apenas uma máquina;

(b) Cada tarefa j possui um tempo de processamento p_{jk} que depende da máquina k na qual será alocada. Por esta característica, as máquinas são ditas não-relacionadas;

(c) Existem tempos de preparação entre as tarefas, S_{ijk}, em que k representa a máquina cujas tarefas i e j serão processadas, nesta ordem. Tais tempos de preparação são dependentes da sequência e da máquina. A dependência da sequência é mostrada pela desigualdade $S_{ijk} \neq S_{jik}$ e a dependência da máquina pode ser percebida nesta desigualdade $S_{ijk} \neq S_{ijh}$, onde h é uma outra máquina;

(d) Considera-se, também, o tempo de preparação para processar a primeira tarefa, representado por S_{0jk}, em que 0 indica uma tarefa fictícia com tempo de processamento nulo ($p_{0k} = 0$).

O objetivo deste problema é encontrar um sequenciamento das n tarefas nas m máquinas de forma a minimizar o tempo máximo de conclusão do sequenciamento. Tempo este, que será consumido pela máquina que concluir suas tarefas por último, também chamada de máquina gargalo. Esse tempo máximo de conclusão é o chamado *makespan*, denotado por $C_{\max}$. Pelas características citadas, o UPMSPST é representado por $R_M \mid S_{ijk} \mid C_{\max}$ (Graham et al., 1979). Nesta representação R_M indica as máquinas não-relacionadas, S_{ijk} os tempos de preparação e $C_{\max}$ o *makespan*.

O PMSP com máquinas idênticas e sem tempos de preparação ($P_M||C_{\max}$) pertence à classe NP-difícil, mesmo quando se tem duas máquinas (Karp, 1972; Garey

e Johnson, 1979). Como o UPMSPST é uma generalização do $P_M||C_{\max}$, logo ele também pertence à classe NP-difícil.

Para melhor compreensão do problema, toma-se uma instância, do mesmo, que contém sete tarefas e duas máquinas. A Tabela 2.1 contém os tempos de processamento dessas tarefas nas duas máquinas, enquanto nas Tabelas 2.2 e 2.3 estão contidos os tempos de preparação dessas tarefas nessas máquinas.

Tabela 2.1: Tempos de processamento nas máquinas M1 e M2

	M1	M2
1	20	4
2	25	21
3	28	14
4	17	32
5	43	38
6	9	23
7	58	52

Na primeira coluna da Tabela 2.1 estão representadas as tarefas. Nas colunas M1 e M2 estão presentes os tempos de processamento de cada tarefa nas máquinas M1 e M2, respectivamente.

Tabela 2.2: Tempos de preparação na máquina M1

M1	1	2	3	4	5	6	7
1	2	1	8	1	3	9	6
2	4	7	6	3	7	8	4
3	7	3	4	2	3	5	3
4	3	8	3	5	5	2	2
5	8	3	7	9	6	5	7
6	8	8	1	2	2	1	9
7	1	4	5	2	3	5	1

Na primeira linha e coluna das Tabelas 2.2 e 2.3 estão representadas as tarefas. A diagonal principal mostra os tempos de preparação inicial para cada tarefa. Como um exemplo, na linha 3 e coluna 3 da Tabela 2.3, tem-se o tempo de preparação inicial para processar a tarefa 3 na máquina M2, $S_{032} = 4$. Os outros valores das tabelas representam os tempos de preparação entre as tarefas, sendo que a linha indica que a tarefa já foi processada e a coluna indica que a tarefa será processada. Desse modo, na linha 5 e coluna 4 da Tabela 2.2 está o tempo de preparação necessário para poder processar a tarefa 4 após ter processado a tarefa 5 na máquina M1, tempo este representado por $S_{541} = 9$.

A Figura 2.1 ilustra um possível sequenciamento para a instância apresentada. Observa-se que na máquina M1 há o sequenciamento das tarefas 2, 1 e 7, nesta ordem. O sequenciamento das tarefas 5, 4, 6 e 3, nesta ordem, na máquina M2 também

Tabela 2.3: Tempos de preparação na máquina M2

M2	1	2	3	4	5	6	7
1	3	4	6	5	9	3	2
2	1	2	6	2	7	7	5
3	2	6	4	6	8	1	4
4	5	7	8	3	2	5	6
5	7	9	5	7	6	4	8
6	9	3	5	4	9	8	3
7	3	2	6	1	5	6	7

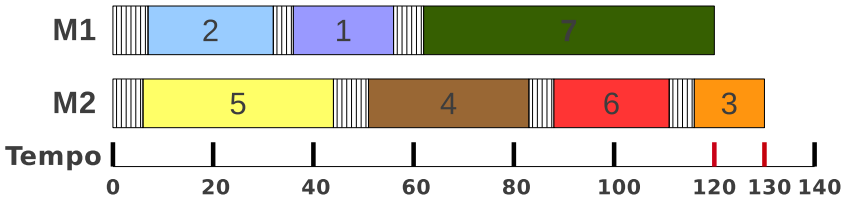

Figura 2.1: Exemplo de um possível sequenciamento

é percebido por esta figura. As partes hachuradas da figura representam os tempos de preparação entre as tarefas e as partes coloridas os tempos de processamento. Na linha abaixo do sequenciamento há a marcação do tempo, em qual os tempos marcados em vermelho representam os tempos de conclusão de cada máquina.

Como a tarefa 6 está alocada à máquina M2 seu tempo de processamento p_{62} será 23. Sua predecessora e sua sucessora são as tarefas 4 e 3, respectivamente. Assim, neste exemplo, são computados os tempos $S_{462} = 5$ e $S_{632} = 5$.

Desta forma, pode-se calcular o tempo de conclusão da máquina M1 como:

$$\begin{aligned} M1 &= S_{021} + p_{21} + S_{211} + p_{11} + S_{171} + p_{71} \\ M1 &= 7 + 25 + 4 + 20 + 6 + 58 = 120 \end{aligned}$$

De modo equivalente também é calculado o tempo de conclusão da máquina M2:

$$\begin{aligned} M2 &= S_{052} + p_{52} + S_{542} + p_{42} + S_{462} + p_{62} + S_{632} + p_{32} \\ M2 &= 6 + 38 + 7 + 32 + 5 + 23 + 5 + 14 = 130 \end{aligned}$$

Após o cálculo dos tempos de conclusão das máquinas M1 e M2, percebe-se que a máquina M2 é a máquina gargalo, ou seja, é a máquina que possui o maior tempo de conclusão, o *makespan*.

Capítulo 3

Revisão Bibliográfica

Neste Capítulo são descritas as principais referências bibliográficas relacionadas ao problema abordado (Seção 3.1), bem como as formulações de programação matemática para o UPMSPST encontradas na literatura (Seção 3.2). Também são expostas as heurísticas (Seção 3.3), metaheurísticas (Seção 3.4) e uma técnica que faz um balanço entre intensificação e diversificação (Seção 3.5), que serviram de inspiração para o desenvolvimento deste trabalho.

3.1 Trabalhos Relacionados

Na literatura, são encontradas poucas referências relacionadas ao UPMSPST. Com a intenção de facilitar a leitura da revisão, primeiramente, na Seção 3.1.1, são listados os trabalhos que tratam do problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência e não das máquinas (aqui nomeado UPMSPST sem tempos de preparação dependentes das máquinas). Em seguida, são revisados os trabalhos que tratam, de fato, do UPMSPST (Seção 3.1.2).

3.1.1 UPMSPST sem Tempos de Preparação Dependentes das Máquinas

Weng et al. (2001) propõem heurísticas construtivas com o intuito de minimizar o tempo total de atraso ponderado. As heurísticas construtivas são baseadas em critérios gulosos. Os critérios são os seguintes: a) alocar as tarefas nas máquinas de menor custo; b) alocar as tarefas nas máquinas em que elas tiverem o menor tempo de processamento; c) alocar as tarefas que tiverem a menor proporção entre o tempo de processamento somado com o tempo de preparação e dividido pelo peso. Foram implementadas sete heurísticas baseadas nesses critérios e combinação deles. A heurística que se mostrou a mais eficiente é aquela que, a cada passo, aloca a tarefa que possui a menor proporção entre a soma do tempo de processamento com o tempo de preparação dividido pelo peso.

Kim et al. (2002) utilizam a metaheurística *Simulated Annealing* objetivando a minimização do tempo total de atraso, no problema de sequenciamento em lotes. Na metaheurística proposta, é utilizada a ideia de perturbações nas soluções, per-

turbações estas baseadas em estruturas de vizinhança. Nessa abordagem, cada perturbação é caracterizada por realizar buscas locais em cada estrutura de vizinhança proposta, ao final é retornada a melhor solução encontrada. Nestas buscas locais, são utilizados movimentos de troca entre os lotes, inserções entre os lotes, fusões dos lotes, separações dos lotes, trocas de itens e inserções de itens. A metaheurística desenvolvida foi comparada a um método de descida que utiliza a mesma ideia de perturbação e os movimentos propostos, bem como comparada a uma versão convencional do *Simulated Annealing*, que realiza movimentos de trocas e inserções de itens. Após a análise dos resultados, a versão que implementou o *Simulated Annealing* com a ideia de perturbações nas soluções foi considerada a melhor pelos autores.

Já Kim et al. (2003) tratam um problema que incorpora datas de entrega iguais, sendo implementadas três heurísticas construtivas e a metaheurística *Simulated Annealing* a fim de se obter o menor atraso ponderado. As heurísticas construtivas são baseadas na data de entrega mais cedo ponderada, no menor tempo de processamento ponderado e na heurística de sequenciamento em lotes de dois níveis. Os testes eram relativos a uma fábrica de semicondutores compostos e mostraram que a heurística de sequenciamento em lotes de dois níveis e a metaheurística *Simulated Annealing* obtiveram os melhores resultados.

Logendran et al. (2007) utilizam quatro regras de despacho para gerar soluções iniciais e uma Busca Tabu, objetivando minimizar o tempo total de atraso ponderado e ainda considerando lançamento dinâmico das tarefas e disponibilidade dinâmica das máquinas. As regras de despacho são a data de entrega mais cedo, menor atraso ponderado, a menor proporção entre a data de entrega e o peso e a menor proporção entre a data de entrega dividida pela soma do tempo de preparação com o tempo de lançamento vezes o peso. Também são desenvolvidas e testadas seis variações da Busca Tabu, baseadas em conceitos de memória de curto e longo prazo, tamanho da lista tabu e reinício da busca. Ao final, os autores concluem que a versão da Busca Tabu com memória de curto prazo e lista tabu fixa deve ser utilizada para problemas menores, a versão com frequência mínima e lista tabu fixa para problemas de tamanho médio e a versão com lista tabu variável e frequência mínima para problemas maiores.

Um algoritmo de *Branch-and-Price* é desenvolvido por Pereira Lopes e de Carvalho (2007) para um problema em que são incluídas datas de disponibilidade para as máquinas e datas de lançamento para as tarefas. É proposto um novo método de aceleração de geração de colunas, chamado de *primal box*, bem como uma regra de seleção de variáveis a serem ramificadas, reduzindo significativamente a quantidade de nós a serem explorados. Os resultados mostraram que o algoritmo criado é capaz de resolver problemas de larga escala em sua otimalidade em tempo aceitável.

Ainda excluindo os tempos de preparação dependentes das máquinas encontra-se um algoritmo genético adaptativo proposto por Randall e Kurz (2007) para um problema envolvendo datas de entregas, tendo como objetivo minimizar o tempo total de atraso ponderado. Os autores representam os indivíduos com chaves randômicas. A população inicial é gerada aleatoriamente. São utilizados quatro operadores de cruzamento: um ponto de corte, dois pontos de corte, uniforme e uniforme parametrizado. As aplicações destes operadores são adaptadas de acordo com a evolução, sendo que aqueles que geram melhores indivíduos são escolhidos. Em cada geração,

20% da população é gerada pela reprodução elitista (permanecem os 20% melhores indivíduos da população), 79% pelo cruzamento (realiza cruzamento a partir de 2 pais escolhidos aleatoriamente) e 1% por imigração (gera um indivíduo aleatoriamente para ser incluído na população).

3.1.2 UPMSPST

de Paula et al. (2007) estudam a implementação do método *Variable Neighborhood Search* (VNS) para resolver problemas em máquinas não-relacionadas e também idênticas, porém este problema difere do problema abordado (UPMSPST), pois são impostas datas de entrega para cada tarefa e penalidades caso haja atraso nestas datas de entrega. O objetivo considerado pelos autores é o de minimizar a soma do *makespan* com os atrasos ponderados.

Dentre os trabalhos que abordam o UPMSPST são encontradas referências mais recentes.

Al-Salem (2004) foi o primeiro a abordar o UPMSPST e apresenta uma heurística de particionamento de três fases, chamada *Partitioning Heuristic* (PH). Na primeira fase é utilizada uma heurística construtiva que aloca as tarefas com base na razão do menor tempo de processamento acrescido da média dos tempos de preparação da máquina. A segunda fase é a caracterizada por refinar a solução construída, o que é feito pela heurística *Composite Exchange Heuristic*. Esta heurística consiste em realizar remoções de tarefas da máquina que contém o *makespan* e inserir na máquina com menor tempo de conclusão até que se encontre um ótimo local. Na terceira e última fase cada máquina é tratada como um problema do caixeiro viajante e são utilizadas duas heurísticas para resolver esse problema, a heurística do vizinho mais próximo e a heurística de trocas de pares adjacentes. O autor ainda propôs o cálculo de um limite inferior para o UPMSPST, comparando seus resultados com esse limite, considerado relativamente fraco pelo mesmo.

Rabadi et al. (2006) implementam uma metaheurística para busca randômica priorizada (*Meta-heuristic for Randomized Priority Search* ou Meta-RaPS) para resolver o UPMSPST. Esta metaheurística utiliza uma estratégia que combina heurísticas de construção e refinamento, e utiliza ideias semelhantes às do *Greedy Randomized Adaptive Search Procedure* (GRASP). A heurística de construção, nomeada SAP-SL, consiste em sequenciar a tarefa que possui o menor tempo de processamento ajustado (*Shortest Adjusted Processing time* - SAP), ou seja, o tempo de processamento somado ao tempo de preparação, na máquina que estiver menos sobrecarregada (*Smallest Load* - SL). É incorporada uma aleatoriedade a essa heurística construtiva, semelhante à fase de construção do GRASP, para gerar soluções diferentes. Na fase de refinamento, são aplicadas três buscas locais com movimentos de inserções entre máquinas, troca entre máquinas de pares de tarefas e trocas randômicas na mesma máquina. Os autores também geraram e disponibilizaram, em SchedulingResearch (2005), problemas-teste para o UPMSPST, os quais foram bastante estudados pelos trabalhos que se sucederam.

Helal et al. (2006) apostam na Busca Tabu para resolver o UPMSPST. Na construção da solução inicial é utilizada para função de avaliação o menor tempo médio de processamento. Tal tempo é calculado por meio da simulação de alocações das tarefas em todas as máquinas. A Busca Tabu possui duas fases de perturbações, uma

utilizando movimentos na mesma máquina e a outra utilizando movimentos entre as máquinas. Às instâncias disponíveis em SchedulingResearch (2005) foi aplicado o algoritmo proposto, obtendo bons resultados.

Arnaout et al. (2010) implementam o método de otimização por colônia de formigas (*Ant Colony Optimization* - ACO) para estruturas especiais do problema UPMSPST em que a proporção do número de tarefas e o número de máquinas é grande. O algoritmo é realizado em dois estágios, o primeiro de atribuição de tarefas e o outro de sequenciamento de tarefas. Sendo assim, são utilizados dois feromônios, um para favorecer a alocação de uma tarefa em uma máquina e o outro para favorecer a inserção de uma outra tarefa após a primeira tarefa na mesma máquina. Também são feitas buscas locais por meio de movimentos de inserção de tarefas em outras máquinas e trocas de tarefas entre as máquinas. Os testes foram feitos com as instâncias disponíveis em SchedulingResearch (2005) e a eficiência do algoritmo para as estruturas especiais foi comprovada.

Ying et al. (2010) propõem um método restrito de *Simulated Annealing* (*Restricted Simulated Annealing* - RSA) para resolver o UPMSPST. A ideia deste método é reduzir o esforço computacional da busca eliminando movimentos de tarefas que não serão eficientes. A construção da solução inicial é feita de forma aleatória. A estratégia utilizada no RSA é de usar movimentos de inserções e trocas de tarefas, porém somente executando movimentos que irão contribuir para uma solução melhor. Segundo os autores, a partir de três proposições é possível eliminar movimentos ineficientes. A primeira é que se o número de máquinas gargalo é maior que dois, qualquer troca entre duas tarefas não irá melhorar a solução. A segunda diz que se o número de máquinas gargalo for dois, apenas trocando duas tarefas pertencentes a essas máquinas, separadamente, é que pode se ter uma chance de melhorar a solução. A terceira afirma que se existir uma máquina gargalo, apenas trocas com pelo menos uma tarefa dessa máquina terão chances de melhorar a solução. O RSA também foi testado com os problemas-teste disponíveis em SchedulingResearch (2005) e conseguiu melhorar as soluções para as instâncias grandes.

Chang e Chen (2011) desenvolvem e provam um conjunto de propriedades para o UPMSPST, além de implementá-las por meio de um Algoritmo Genético, intitulado GADP, e também pelo *Simulated Annealing*, nomeado SADP. As propriedades abordadas são baseadas em movimentos realizados entre as máquinas e na mesma máquina. De acordo com os autores, ao aplicar tais propriedades chega-se a uma solução próxima da solução ótima. Os algoritmos foram implementados e testados nas instâncias disponíveis em SchedulingResearch (2005) e o GADP se mostrou o mais eficiente, conseguindo melhorar as soluções para instâncias maiores.

Fleszar et al. (2011) apresentam uma hibridização que une o algoritmo *Multi-start*, o *Variable Neighbourhood Descent* (VND) e um modelo de programação matemática. Tal algoritmo híbrido, nomeado MVND, repete os procedimentos de construção e refinamento por dez vezes. A solução inicial é construída de maneira que a escolha das tarefas a serem incluídas na mesma é realizada aleatoriamente. Após escolhida uma tarefa, ela é inserida na posição que contribuirá para o menor tempo de conclusão de uma máquina. Depois de gerada, a solução passa por um refinamento de seu sequenciamento, realizado pelo algoritmo *SeqOpt*. A seguir é aplicada uma busca local por movimentos de trocas e inserções. Em seguida, um modelo de programação inteira mista é utilizado para definir qual outro tipo de movimento

pode ser executado para melhorar a solução. O MVND também foi testado com os problemas-teste disponíveis em SchedulingResearch (2005), mostrando seu poder para resolver o UPMSPST ao resolver instâncias maiores em pouco tempo e obter soluções de qualidade.

Vallada e Ruiz (2011) propõem a resolução do UPMSPST por meio de Algoritmos Genéticos. Nos algoritmos desses autores, a população inicial é criada com um indivíduo pela heurística de múltipla inserção Kurz e Askin (2001) e o resto da população é gerado aleatoriamente. Depois de criados, são aplicadas buscas locais em todos os indivíduos. A seleção para o cruzamento é realizada por torneio n-ário. Nos cruzamentos são realizados procedimentos de buscas locais limitadas. Buscas locais baseadas em movimentos de múltipla inserção entre máquinas também são aplicadas a todos os indivíduos. A seleção para a sobrevivência é feita pela estratégia estacionária, incluindo indivíduos originais na população, desde que sejam melhores que os piores. É importante ressaltar que neste trabalho, os autores não consideram tempos de preparação para as primeiras tarefas alocadas nas máquinas, sendo, portanto, uma característica particular deste UPMSPST. Deste modo, os autores geraram problemas-teste para tal problema e disponibilizaram-os em SOA (2011). Os autores desenvolveram algoritmos baseados nos trabalhos de Kurz e Askin (2001) e Rabadi et al. (2006). Além disso, foram desenvolvidos dois algoritmos genéticos, nomeados GA1 e GA2, que diferiam entre si apenas com relação aos parâmetros adotados (tamanho da população, probabilidade de cruzamento, probabilidade de mutação, probabilidade de aplicação da busca local e pressão seletiva). O algoritmo que obteve os melhores resultados para tais instâncias foi o GA2, com os seguintes parâmetros: tamanho da população = 80, probabilidade de cruzamento = 50%, probabilidade de mutação = 50%, probabilidade de aplicação da busca local = 100% e número de indivíduos participantes do torneio = 8.

3.2 Formulação de Programação Matemática

Uma característica importante dos métodos exatos é a de encontrar soluções ótimas para problemas combinatórios. Por isso, a utilização desses métodos se torna fundamental para que se validem os métodos heurísticos utilizados. Tal validação pode ser feita aplicando-os em problemas de pequena dimensão.

Neste Capítulo são apresentadas duas formulações de programação matemática para representar o UPMSPST, sendo uma de Vallada e Ruiz (2011) (Seção 3.2.1) e outra, com menor número de variáveis e restrições, de Rabadi et al. (2006) (Seção 3.2.2).

3.2.1 Formulação de Vallada e Ruiz (2011)

Vallada e Ruiz (2011) propuseram um modelo de programação inteira mista (PIM) para resolver o UPMSPST, adaptando o modelo encontrado em Guinet (1993). As seguintes notações são utilizadas por este modelo:

- $M = \{1, ..., m\} \rightarrow$ é o conjunto de máquinas, sendo m o número de máquinas;

- $N = \{1, ..., n\} \rightarrow$ é o conjunto de tarefas, com n representando o número de tarefas;
- $\{0\} \cup N \rightarrow$ é o conjunto de tarefas com a adição da tarefa 0 (fictícia);
- $p_{ij} \rightarrow$ é o tempo de processamento da tarefa j na máquina i;
- $S_{ijk} \rightarrow$ é o tempo de preparação necessário para poder processar a tarefa k se ela for sequenciada após a tarefa j na máquina i;
- $V \rightarrow$ é uma constante suficientemente grande

As variáveis de decisão utilizadas pelo modelo são:

- $x_{ijk} = \rightarrow 1$ se a tarefa j antecede a tarefa k na máquina i e 0 caso contrário;
- $C_{ij} \rightarrow$ é o tempo de conclusão da tarefa j na máquina i;
- $C_{\max} \rightarrow$ é o tempo de conclusão máximo

A seguir é apresentado o modelo PIM:

$$\min \quad C_{\max} \tag{3.1}$$

$$s.a.$$

$$\sum_{i=1}^{m} \sum_{\substack{j=0 \\ j \neq k}}^{n} x_{ijk} = 1 \qquad \forall k \in N \tag{3.2}$$

$$\sum_{i=1}^{m} \sum_{\substack{k=1 \\ j \neq k}}^{n} x_{ijk} \leq 1 \qquad \forall j \in N \tag{3.3}$$

$$\sum_{k=1}^{n} x_{i0k} \leq 1 \qquad \forall\, i \in M \tag{3.4}$$

$$\sum_{\substack{h=0 \\ h \neq k \\ h \neq j}}^{n} x_{ihj} \geq x_{ijk} \qquad \forall j, k \in N, j \neq k, \forall i \in M \tag{3.5}$$

$$C_{ik} + V\,(1 - x_{ijk}) \geq C_{ij} + S_{ijk} + p_{ik} \quad \forall j \in \{0\} \cup N, \forall k \in N, j \neq k, \forall i \in M \tag{3.6}$$

$$C_{i0} = 0 \qquad \forall i \in M \tag{3.7}$$

$$C_{ij} \geq 0 \qquad \forall j \in N, \forall i \in M \tag{3.8}$$

$$C_{\max} \geq C_{ij} \qquad \forall j \in N, \forall i \in M \tag{3.9}$$

$$x_{ijk} \in \{0,\, 1\} \quad \forall j \in \{0\} \cup N, \forall k \in N, j \neq k, \forall i \in M \tag{3.10}$$

O objetivo (3.1) é minimizar o *makespan* ($C_{\max}$). As restrições (3.2) garantem que cada tarefa é alocada em apenas uma máquina e possui apenas um predecessor. Com as restrições (3.3) o número máximo de sucessores de cada tarefa é limitado a 1. Também é limitado a 1, em cada máquina, o número máximo de sucessores das tarefas fictícias nas restrições (3.4). As restrições (3.5) garantem as alocações

corretas das tarefas nas máquinas. Se uma dada tarefa j é processada na máquina i, deve existir uma tarefa antecessora h na mesma máquina. Já as restrições (3.6) servem para controlar os tempos de conclusão das tarefas nas máquinas. Se a tarefa k está alocada após a tarefa j na máquina i ($x_{ijk} = 1$), o tempo de conclusão relacionado, C_{ik}, deve ser maior que o tempo de conclusão para j, C_{ij}, somado ao tempo de preparação entre j e k com o tempo de processamento de k. Caso $x_{ijk} = 0$, a constante V fará com que a restrição seja redundante. As restrições (3.7) e (3.8) definem os tempos de conclusão como 0 para as tarefas fictícias e positivas para as tarefas regulares, respectivamente. As restrições (3.9) definem o tempo máximo de conclusão (*makespan*). Finalmente, as restrições (3.10) definem as variáveis binárias.

3.2.2 Formulação de Rabadi et al. (2006)

A seguir é apresentado o modelo de programação inteira mista (PIM) para resolver o UPMSPST proposto em Rabadi et al. (2006), ao adaptar o modelo de Guinet (1991). Este modelo considera as seguintes notações:

- $M = \{1, ..., m\} \rightarrow$ é o conjunto de máquinas, sendo m o número de máquinas;
- $N = \{1, ..., n\} \rightarrow$ é o conjunto de tarefas, com n representando o número de tarefas;
- $N_0 = N \cup \{0\} \rightarrow$ é o conjunto de tarefas com a adição da tarefa 0 (fictícia);
- $p_{jk} \rightarrow$ é o tempo de processamento da tarefa j na máquina k;
- $S_{ijk} \rightarrow$ é o tempo de preparação necessário para poder processar a tarefa j se ela for sequenciada após a tarefa i na máquina k;
- $S_{0jk} \rightarrow$ é o tempo de preparação necessário para poder processar a tarefa j se ela for a primeira tarefa a ser processada na máquina k;
- $L \rightarrow$ é uma constante suficientemente grande

As seguintes variáveis de decisão são utilizadas no modelo:

- $x_{ijk} = \rightarrow 1$ se a tarefa j está sequenciada após a tarefa i na máquina k e 0 caso contrário;
- $x_{0jk} = \rightarrow 1$ se a tarefa j for a primeira a ser processada na máquina k e 0 caso contrário;
- $x_{i0k} = \rightarrow 1$ se a tarefa i for a última a ser processada na máquina k e 0 caso contrário;
- $C_j \rightarrow$ é o tempo de conclusão da tarefa j;
- $C_{\max} = max_{j \in N}\, C_j \rightarrow$ é o *makespan* da solução

$$\min \quad C_{\max} \tag{3.11}$$

$$s.a.$$

$$0 \leq C_j \leq C_{\max} \qquad \forall j \in N_0 \tag{3.12}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} \sum_{k=1}^{m} x_{ijk} = 1 \qquad \forall j \in N \tag{3.13}$$

$$\sum_{\substack{i=0 \\ i \neq h}}^{n} x_{ihk} = \sum_{\substack{j=0 \\ j \neq h}}^{n} x_{hjk} \qquad \forall h \in N,\ \forall k \in M \tag{3.14}$$

$$C_i + \sum_{k=1}^{m} (S_{ijk} + p_{jk}) x_{ijk} + L \left(\sum_{k=1}^{m} x_{ijk} - 1 \right) \leq C_j \qquad \forall i \in N_0,\ \forall j \in N \tag{3.15}$$

$$\sum_{j=0}^{n} x_{0jk} = 1 \qquad \forall\ k \in M \tag{3.16}$$

$$x_{ijk} \in \{0,\ 1\} \qquad \forall\ i, j \in N_0, \forall k \in M \tag{3.17}$$

O objetivo (3.11) é minimizar o *makespan*. As restrições (3.12) garantem que o *makespan* seja o tempo máximo de conclusão das tarefas. As restrições (3.13) asseguram que cada tarefa será alocada apenas uma vez e processada por apenas uma máquina. As restrições (3.14) garantem que para cada tarefa e para cada máquina, o número de predecessores da tarefa é igual ao número de sucessores da mesma. As restrições (3.15) são utilizadas para calcular os tempos de conclusão de cada tarefa e ainda assegurar que uma tarefa não possa preceder e suceder a mesma tarefa, ou seja, impede a criação de ciclos de processamento. As restrições (3.16) garantem que exatamente uma tarefa está sequenciada como a primeira tarefa de cada máquina. Finalmente, as restrições (3.17) definem os domínios das variáveis.

3.3 Heurísticas

Uma heurística é uma técnica, cuja origem vem da inspiração em processos intuitivos, que realiza busca por uma boa solução consumindo um custo computacional aceitável. Essa técnica, entretanto, não é capaz de garantir a otimalidade da solução encontrada, além de não se ter a garantia da proximidade da solução ótima.

De acordo com Gomes (2003), a maioria dos métodos heurísticos são utilizados na resolução de problemas cuja utilização de métodos exatos se torna inviável. A ideia desses métodos heurísticos é tentar reproduzir, na maioria das vezes, o trabalho que é realizado manualmente.

As heurísticas podem ser classificadas em heurísticas construtivas, descritas na Seção 3.3.1, e heurísticas de refinamento, apresentadas na Seção 3.3.2.

3.3.1 Heurísticas Construtivas

Uma heurística construtiva tem a finalidade de gerar uma solução e faz isso incorporando elemento a elemento na solução. Cada elemento a ser inserido na solução

é escolhido com base na função de avaliação adotada, a qual depende do problema abordado. Geralmente, funções gulosas são utilizadas como guia na ordenação dos elementos candidatos a serem incluídos na solução, calculando o benefício da inserção de cada elemento e inserindo, sempre, o elemento que trará o maior benefício para a solução final.

O Algoritmo 1 ilustra uma construção gulosa para um problema de minimização, cujo parâmetro é a função de avaliação g. Primeiramente deve-se inicializar o conjunto C de elementos candidatos a serem incluídos na solução e a nova solução s. Enquanto existirem elementos candidatos, deve-se procurar o elemento em C que tiver o menor resultado após a aplicação de g. Esse elemento será inserido na solução e removido de C. Ao final se tem uma solução construída com todos os elementos.

Algoritmo 1: Heurística Construtiva Gulosa - Clássico

Entrada: $g(.)$
Saída: Solução s construída
1 Inicialize o conjunto C de elementos candidatos;
2 $s \leftarrow \emptyset$;
3 **enquanto** $(|C| > 0)$ **faça**
4 $\quad g(c^*) = \arg\min\{g(t) \mid t \in C\}$;
5 $\quad s \leftarrow s \cup \{c^*\}$;
6 $\quad C = C \setminus \{c^*\}$;
7 **fim**
8 **Retorne** s;

Uma outra maneira de construir uma solução é de forma aleatória. Sendo assim, ao invés de escolher um elemento com base em uma função de avaliação, deve-se, a cada iteração, sortear um elemento candidato que será inserido na solução. Este método é de fácil implementação, porém as soluções construídas geralmente são bem diferentes e de qualidade muito inferior ao método guloso, que, por sua vez, é caracterizado por ser determinístico e gerar soluções de melhor qualidade.

3.3.2 Heurísticas de Refinamento

As heurísticas de refinamento são métodos cujo objetivo é melhorar soluções já construídas, por meio de modificações em seus elementos. Estas modificações são baseadas em noções de estrutura de vizinhança. Cada vizinho da solução corrente é pesquisado na tentativa de encontrar o melhor vizinho, ou seja, a melhor solução com base na estrutura de vizinhança adotada.

Seja V uma função, dependente da estrutura do problema tratado, que associa a cada solução $s \in S$, sua vizinhança $V(s) \subseteq S$. Cada solução $s' \in V(s)$ é chamada de vizinho de s. Define-se *movimento* como uma modificação m que transforma uma solução s em outra, s', que esteja em sua vizinhança. Esta operação é representada por $s' \leftarrow s \oplus m$.

As heurísticas de refinamento mais conhecidas são o método da descida e método da subida, os quais podem ser realizados de forma completa (Seção 3.3.2.1) ou com a estratégia de primeira melhora (Seção 3.3.2.2).

3.3.2.1 Método da Descida/Subida Completa

O Método da Descida/Subida Completa parte de uma solução e busca o melhor vizinho desta solução com base em uma estrutura de vizinhança. Isto é feito movendo sempre para o vizinho que representa uma melhora na função de avaliação. Devido ao fato desse método avaliar todos os vizinhos, o mesmo é caracterizado por utilizar a estratégia *Best Improvement*. Se o melhor vizinho for um ótimo local, o método é finalizado; caso contrário, a busca continua. A desvantagem deste método é que ele fica preso no primeiro ótimo local encontrado.

Diz-se Método da Descida quando o problema em questão é de minimização e Método da Subida quando o problema tratado é de maximização.

No Algoritmo 2 encontra-se o pseudocódigo para o Método da Descida Completa que recebe como parâmetros uma função de avaliação f, uma estrutura de vizinhança V e uma solução s.

Algoritmo 2: Método da Descida Completo

Entrada: $(f(.), V(.), s)$
Saída: Solução s refinada
1 $N \leftarrow \{s' \in V(s) \mid f(s') < f(s)\}$;
2 **enquanto** $(|N| > 0)$ **faça**
3 | Selecione $s' \in N$, onde $s' = \arg\min\{f(s) \mid s \in N\}$;
4 | $s \leftarrow s'$;
5 | $N \leftarrow \{s' \in V(s) \mid f(s') < f(s)\}$;
6 **fim**
7 **Retorne** s;

3.3.2.2 Método da Descida/Subida de Primeira Melhora

Muitas vezes, analisar toda a estrutura de vizinhança requer um custo computacional elevado. O Método da Descida/Subida de Primeira Melhora surgiu para evitar esse esforço computacional. Neste método, caracterizado por utilizar a estratégia denominada *First Improvement*, a exploração da estrutura de vizinhança é interrompida quando se encontra um vizinho melhor, não necessariamente o melhor. Sendo assim, apenas no pior caso ocorre a análise de toda a estrutura de vizinhança. A desvantagem desse método, tal como o anterior, é que ele fica preso ao primeiro ótimo local encontrado.

3.4 Metaheurísticas

Como dito anteriormente, as heurísticas de refinamento quando encontram um ótimo local terminam seu funcionamento. Porém, esse ótimo local encontrado pode estar bem distante do ótimo global. Visando a uma exploração maior do espaço de busca e a fuga de ótimos locais que não são ótimos globais, surgiram as chamadas metaheurísticas. Pode-se dizer que as metaheurísticas são métodos que visam a obtenção de boas soluções, eventualmente ótimas, e que consistem em aplicações de heurísticas

subordinadas, as quais são modeladas de acordo com o problema abordado (Ribeiro, 1996).

O que diferencia as metaheurísticas é a forma utilizada para explorar o espaço de busca e, consequentemente, como escapar das armadilhas dos ótimos locais. Basicamente, as metaheurísticas são dividas em busca local e busca populacional.

Nas metaheurísticas baseadas em busca local, a exploração do espaço de busca é realizada por meio de movimentos que são aplicados a cada passo sobre a solução corrente para gerar uma outra solução promissora em sua vizinhança. *Iterated Local Search* (Seção 3.4.1), *Variable Neighborhood Search* (Seção 3.4.2) e *Greedy Randomized Adaptive Search* (Seção 3.4.3) são exemplos de métodos que se enquadram nesta categoria.

Já as metaheurísticas baseadas em busca populacional mantêm um conjunto de boas soluções e recombinam-as com o objetivo de produzir melhores soluções. Um exemplo clássico de uma metaheurística desta categoria é o Algoritmo Genético.

Nas próximas subseções são descritas as metaheurísticas utilizadas durante este trabalho.

3.4.1 *Iterated Local Search*

O *Iterated Local Search* – ILS (Lourenço et al., 2003) é um método de busca local que tem como fundamento principal a aplicação de perturbações nas soluções. Ao fazer isso, a intenção é a de impedir a estagnação da busca em um ótimo local que é consideravelmente pior que o ótimo global. Em uma perturbação, o número de componentes modificados de uma solução é chamado de força de perturbação. As perturbações devem ser fortes o suficiente para que elas possam conduzir a busca para regiões diferentes do espaço de soluções, mas também devem ser fracas o suficiente para evitar reinícios aleatórios e poder preservar características do ótimo local corrente.

É importante ressaltar que, segundo Lourenço et al. (2003), uma boa perturbação transforma uma solução excelente em um excelente ponto de partida para uma busca local. Ainda segundo eles, uma maneira sofisticada de gerar boas perturbações é otimizar uma subparte do problema.

O Algoritmo 3 representa o funcionamento do ILS. Inicialmente deve-se implementar um método que gere uma solução inicial (linha 1). A esta solução gerada aplica-se uma busca local (linha 2). A solução resultante desta busca será a solução corrente. A seguir, inicia-se a parte iterativa do algoritmo, na qual a solução corrente sofre, primeiramente, uma perturbação (linha 4) e logo após passa por um processo de busca local (linha 5). A solução proveniente da perturbação e da busca local passará por um critério de avaliação (linha 6), onde acontecerá a atualização da melhor solução e da solução corrente. Ao final do ILS, a melhor solução encontrada é retornada (linha 8).

O critério de aceitação mais comumente usado é o de aceitar uma solução que melhore a solução corrente. As perturbações são aumentadas à medida que não ocorram melhoramentos na solução corrente. A variável *histórico* armazena informações de quando essas perturbações devem ser aumentadas ou diminuídas.

Algoritmo 3: Iterated Local Search - Clássico

Entrada: critério de parada, função de avaliação
Saída: Solução s refinada

```
1 s_0 ← GeraSolucaoInicial();
2 s ← BuscaLocal(s_0);
3 enquanto (critério de parada não satisfeito) faça
4 |   s' ← Perturbacao(s, histórico);
5 |   s'' ← BuscaLocal(s');
6 |   s ← CriterioAceitacao(s, s'');
7 fim
8 Retorne s;
```

3.4.2 *Variable Neighborhood Descent*

O método *Variable Neighborhood Descent* – VND (Mladenovic e Hansen, 1997) tem como característica a exploração do espaço de soluções através de trocas sistemáticas de estruturas de vizinhanças. Os autores se inspiraram em três princípios ao idealizarem este método:

- Um ótimo local com relação à uma estrutura de vizinhança não é necessariamente um ótimo local relativo à outra estrutura de vizinhança;
- Um ótimo global é um ótimo local com relação a todas as possíveis estruturas de vizinhanças;
- Para muitos problemas, ótimos locais com relação à uma ou mais estruturas de vizinhanças são relativamente próximos.

Ainda segundo os autores, uma observação importante e empírica é a de que um ótimo local geralmente contém informações sobre o ótimo global. Essas informações podem ser várias variáveis com o mesmo valor, em relação ao ótimo global. Contudo, não se pode saber quais variáveis são essas, o que induz investigações sistemáticas da estrutura de vizinhança deste ótimo local até que se encontre uma solução melhor.

No Algoritmo 4 está representado o pseudocódigo do método VND aplicado a um problema de minimização. Os dados de entrada são uma função de avaliação f, um conjunto de vizinhanças V, um número de estruturas de vizinhanças diferentes r e uma solução s. A cada iteração do VND é realizada uma busca local em uma estrutura de vizinhança, começando pela primeira. Ao fim dessa busca se tem o melhor vizinho da solução corrente em relação a esta estrutura de vizinhança. Se este melhor vizinho não for melhor do que a solução corrente, então uma nova busca local na próxima estrutura de vizinhança é realizada. Caso esse melhor vizinho seja melhor do que a solução corrente, então a solução corrente passa a ser esse melhor vizinho e a busca local é reiniciada na primeira estrutura de vizinhança. O método termina quando não é encontrada melhora em nenhuma das vizinhanças adotadas e retorna a melhor solução obtida durante sua execução, ou seja, a melhor solução com relação a todas as estruturas de vizinhanças.

A versão clássica do VND considera as buscas em estruturas de vizinhanças seguindo uma ordem pré-estabelecida. Entretanto, trabalhos recentes comprovaram

Algoritmo 4: Variable Neighborhood Descent - Clássico

Entrada: $f(.), V(.), r, s$
Saída: Solução s refinada
$k \leftarrow 1$; /* Estrutura de vizinhança corrente */
enquanto $(k \leq r)$ **faça**
  Encontre o melhor vizinho $s' \in V^{(k)}(s)$;
  se $(f(s') < f(s))$ **então**
    $s \leftarrow s'$;
    $k \leftarrow 1$
  fim
  senão
    $k \leftarrow k + 1$
  fim
fim
Retorne s;

a eficiência de uma abordagem que estabelece ordens aleatórias de exploração das estruturas de vizinhanças, produzindo uma melhor diversificação da busca. O uso bem sucedido de tal estratégia, denominada *Random Variable Neighborhood Descent* (RVND), é relatado em Souza et al. (2010) e Subramanian et al. (2010).

3.4.3 *Greedy Randomized Adaptive Search Procedure*

O método *Greedy Randomized Adaptive Search Procedure* – GRASP (Feo e Resende, 1995) é um processo iterativo constituído de duas fases, uma fase de construção e outra de busca local. A fase de construção é responsável por gerar uma solução viável iterativamente, elemento a elemento. A fase de busca local consiste em explorar a vizinhança da solução criada até encontrar um ótimo local.

A descrição do funcionamento de um GRASP clássico está representada pelo Algoritmo 5. Os dados de entrada são uma função de avaliação f, uma função adaptativa gulosa g, uma vizinhança V e um parâmetro α.

Algoritmo 5: *Greedy Randomized Adaptive Search* - Clássico

Entrada: $f(.)$, $g(.)$, $V(.)$, α
Saída: Solução s refinada
enquanto *(critério de parada não satisfeito)* **faça**
  $s \leftarrow ConstrucaoGRASP(g(.), \alpha)$;
  $s' \leftarrow BuscaLocal(f(.), V(.), s)$;
  $AtualizaMelhor(s', s^\star)$; /* $s^\star$ é a melhor solução encontrada */
fim
Retorne $s^\star$;

A primeira fase do GRASP é a fase construtiva, que tem por objetivo aliar características positivas oriundas tanto das construções gulosas quanto das construções aleatórias. Uma característica positiva da construção gulosa é que elas normalmente

geram soluções de boa qualidade, próximas a ótimos locais, favorecendo a convergência mais rápida a um ótimo local na aplicação da fase de refinamento. Já a construção aleatória tem a vantagem de gerar soluções diferentes a cada execução; assim, ao gerar soluções que estão localizadas em regiões distintas no espaço de busca, possibilita-se uma exploração maior deste espaço.

O Algoritmo 6 descreve a fase de construção GRASP, que recebe como parâmetros uma função adaptativa gulosa g e α. A cada iteração, uma lista de candidatos (LC) que contém os elementos candidatos a serem incluídos na solução é construída. Nesta lista, os elementos estarão ordenados de acordo com a função adaptativa g. A partir da LC é construída uma outra lista, a Lista Restrita de Candidatos (LRC), que é formada pelos elementos mais bem qualificados de LC. Em seguida, um elemento é escolhido aleatoriamente de LRC e incluído na solução. Ao final deste procedimento se tem uma solução de fato, isto é, composta de todos os elementos.

O tamanho da LRC é controlado pela equação $g(t) \leq g_{\min} + \alpha(g_{\max} - g_{\min})$, sendo $g_{\min}$ o menor valor obtido da aplicação de g aos elementos, $g_{\max}$ o maior valor obtido da aplicação de g aos elementos e $\alpha \in [0, 1]$ um parâmetro que controla o quão gulosa ou aleatória será a solução. O parâmetro é dito controlador da gulosidade e aleatoriedade pois, se $\alpha = 0$, a LRC será composta por apenas um elemento, o elemento correspondente a $g_{\min}$ após a aplicação de g, logo as soluções geradas serão puramente gulosas. Contudo se $\alpha = 1$, a LRC será composta de todos os elementos, caracterizando a geração de soluções totalmente aleatórias.

Algoritmo 6: Fase de Construção GRASP - Clássica

Entrada: $g(.)$, α
Saída: Solução s construída

```
s ← ∅;
Inicialize a lista de candidatos LC;
enquanto (|LC| > 0) faça
    g_min ← min{g(t) | t ∈ LC};
    g_max ← max{g(t) | t ∈ LC};
    LRC ← {t ∈ LC | g(t) ≤ g_min + α(g_max − g_min)};
    Aleatoriamente, selecione um elemento t ∈ LRC;
    s ← s ∪ {t};
    Atualize a lista de candidatos LC;
fim
Retorne s;
```

A segunda fase do GRASP é a fase de busca local, cuja função é de refinar a solução gerada pela fase construtiva. A eficiência desta fase está proporcionalmente relacionada com a qualidade da solução gerada pela primeira fase. Quanto melhor for a solução gerada, mais rápida será a convergência para um ótimo local. E, consequentemente, quanto pior for a solução, mais lenta será a convergência para um ótimo local.

Deve-se evidenciar que a cada iteração uma solução diferente é construída, o que geralmente implica na obtenção de ótimos locais diferentes. O melhor ótimo local encontrado é retornado ao final do GRASP.

3.5 *Path Relinking*

A estratégia *Path Relinking* (PR), proposta por Glover (1996), procura fazer um balanceamento entre intensificação e diversificação da busca. Seu objetivo é explorar trajetórias que interligam soluções de alta qualidade.

Tomando-se duas soluções, uma denominada solução base e a outra chamada de solução guia, a ideia do PR é buscar melhores soluções por meio da construção de um caminho da solução base até a solução guia. Este caminho é construído por meio de movimentos que introduzem atributos da solução guia na solução base.

Essas soluções de alta qualidade são armazenadas em um conjunto com tamanho limitado de soluções elite. Caso este conjunto esteja vazio, a solução será inserida nele. Após a inserção de uma solução, para que a próxima solução encontrada seja incluída ao conjunto, alguma de duas regras deve ser obedecida. Uma solução é incluída no conjunto elite se ela for melhor do que a melhor solução já presente no mesmo. Também é incluída no conjunto elite a solução que for melhor do que a pior solução do conjunto, mas que satisfaça a um nível mínimo de diversidade das outras soluções elite. O objetivo dessa segunda regra é de evitar um conjunto elite com soluções muito parecidas. Ao preencher o conjunto elite com o número de soluções desejado, para que uma solução seja inserida no conjunto, a pior solução do conjunto deve ser excluída.

O Algoritmo 7 descreve o funcionamento do PR, recebendo como entrada a solução base s_{base}, a solução guia s_{guia} e a função de avaliação f. Inicialmente deve-se definir qual tipo de atributo será considerado e gerar um conjunto Δ que contém os atributos da solução base que se diferem da solução guia. Até que todos os atributos da solução guia sejam iguais aos da solução base, deve-se inserir os atributos contidos em Δ na solução base e calcular o menor resultado de f, movimento definido como l^*. Em seguida, remove-se l^* de Δ e aplica-se o movimento na solução base, atualizando a melhor solução encontrada. Quando $|\Delta| = 0$, a solução base será igual à solução guia. Ao final, aplica-se uma busca local na melhor solução encontrada durante o processo e o resultado dessa busca local é retornado.

Algoritmo 7: Path Relinking - Clássico

Entrada: s_{base}, s_{guia}, $f()$
Saída: Melhor solução $s^\star$ encontrada

1 Inicialize Δ com todos os atributos de s_{guia};
2 $s' \leftarrow \arg\min\{f(s_{base}), f(s_{guia})\}$;
3 $x \leftarrow s_{base}$;
4 **enquanto** $(|\Delta| > 1)$ **faça**
5 $\quad l^* \leftarrow \arg\min\{f(x \oplus l) : l \in \Delta\}$;
6 $\quad \Delta \leftarrow \Delta \setminus \{l^*\}$;
7 $\quad x \leftarrow x \oplus l^*$;
8 $\quad$ *AtualizaMelhor*(x, s') ; `/* s' é a melhor solução encontrada */`
9 **fim**
10 $s^\star \leftarrow$ *BuscaLocal*(s');
11 **Retorne** $s^\star$;

De acordo com Rossetti (2003), o PR pode ser aplicado como uma estratégia de

pós-otimização entre todos os pares de soluções pertencentes ao conjunto elite ou pode ser aplicado como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local. Contudo, a aplicação do PR como uma estratégia de intensificação a cada ótimo local se mostrou mais eficaz do que como uma estratégia de pós-otimização (Rossetti, 2003).

Diversas alternativas estão sendo utilizadas e combinadas em implementações recentes do PR, entre as quais destacamos:

- *periodical relinking:* PR não é aplicado sistematicamente, apenas periodicamente;
- *forward relinking:* PR é aplicado com a solução base sendo a pior solução e a solução guia sendo a melhor;
- *backward relinking:* PR é aplicado com a solução base sendo a melhor solução e a solução guia sendo a pior;
- *backward and forward relinking:* duas estratégias são utilizadas, primeiro aplica-se o *backward relinking* e depois aplica-se o *forward relinking*;
- *mixed relinking:* dois caminhos são simultaneamente explorados, o primeiro proveniente da solução base como sendo a pior solução e o segundo proveniente da solução base como sendo a melhor solução, até que esses caminhos se encontrem em uma solução intermediária equidistante da solução base e da solução guia;
- *randomized relinking:* ao invés de selecionar o melhor movimento ainda não selecionado, aleatoriamente selecione um movimento dentre os mais promissores do caminho que está sendo investigado;
- *truncated relinking:* não se explora a trajetória completa entre a solução base até a solução guia, apenas parte dela.

A implementação do PR com a estratégia *backward relinking* foi considerada eficiente por Ribeiro et al. (2002). Entretanto, recentes trabalhos (Resende e Ribeiro, 2005, 2010; Ribeiro e Resende, 2011; Frinhani, 2011; Nogueira, 2011) mostraram a eficácia da estratégia *mixed relinking*, idealizada por Glover (1996).

O Algoritmo 8 mostra o pseudocódigo da estratégia *mixed relinking*.

A diferença deste algoritmo para o Algoritmo 7 são as linhas 9, 10 e 11. Nas linhas 9 e 10 as funções das soluções guia e base são trocadas. Já na linha 11 é calculado um novo Δ com os atributos da solução base que se diferem da solução guia.

Algoritmo 8: Mixed Path Relinking - Clássico

Entrada: s_{base}, s_{guia}, $f()$
Saída: Melhor solução $s^\star$ encontrada

Inicialize Δ com todos os atributos de s_{guia};
$s' \leftarrow \arg\min\{f(s_{base}), f(s_{guia})\}$;
$x \leftarrow s_{base}$;
enquanto $(|\Delta| > 1)$ **faça**
    $l^* \leftarrow \arg\min\{f(x \oplus l) : l \in \Delta\}$;
    $\Delta \leftarrow \Delta \setminus \{l^*\}$;
    $x \leftarrow x \oplus l^*$;
    $AtualizaMelhor(x, s')$; `/* s' é a melhor solução encontrada */`
    $s_{base} \leftarrow s_{guia}$;
    $s_{guia} \leftarrow x$;
    Compute Δ com todos os atributos de s_{guia} ;
fim
$s^\star \leftarrow BuscaLocal(s')$;
Retorne $s^\star$;

Capítulo 4

Metodologia

Neste Capítulo são apresentados os métodos desenvolvidos para a resolução do UPMSPST. Inicialmente, na Seção 4.1, é mostrado como uma solução para o problema é representada. A seguir, na Seção 4.2, está descrito o método de avaliação de uma solução para o UPMSPST. Em seguida, são detalhados os algoritmos propostos para resolver o UPMSPST (Seção 4.3). Finalizando, os módulos que os algoritmos utilizam são apresentados na Seção 4.4.

4.1 Representação da Solução

Uma solução s do UPMSPST é representada como um vetor de listas. Em tal representação se tem um vetor v cujo tamanho é o número de máquinas, m, e cada posição desse vetor contém um número que representa uma máquina. O sequenciamento das tarefas em cada máquina, por sua vez, é representado por uma lista de números, em que cada número representa as tarefas. Para melhor compreensão desta representação tem-se como exemplo a Figura 4.1, na qual s representa o sequenciamento visto na Figura 2.1.

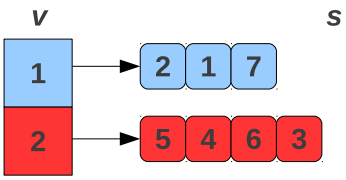

Figura 4.1: Representação como vetor de listas do UPMSPST

4.2 Avaliação de uma Solução

Uma solução s tem como valor de avaliação o tempo máximo de conclusão do sequenciamento, ou seja, o *makespan*, definido por $C_{\max}$.

Entretanto, utilizar somente o *makespan* como critério de comparação de soluções pode fazer com que aperfeiçoamentos em outras máquinas, que não a máquina representativa do $C_{\max}$, sejam rejeitados durante o processo de busca. Sendo assim,

em algum momento é utilizada a soma dos tempos de conclusão das máquinas como critério de comparação das soluções. Este critério pode ser visto na Seção 4.4.9.2.

4.3 Algoritmos propostos

Objetivando a resolução do UPMSPST, são propostos 3 algoritmos: IVP, GIVP e GIVMP. O primeiro (Seção 4.3.1) é um algoritmo que combina os procedimentos heurísticos ILS, VND e PR. O segundo (Seção 4.3.2) é um aperfeiçoamento do primeiro que inclui um módulo de construção GRASP. O terceiro (Seção 4.3.3) difere do segundo no módulo de PR, pois considera a estratégia *mixed relinking* ao invés da estratégia *backward relinking*, difere também no módulo do VND quando não define uma ordem de processamento das buscas locais, e também inclui um módulo de programação matemática que é aplicado na máquina gargalo, isto é, aquela que define o *makespan*. Os algoritmos são, a seguir, apresentados.

4.3.1 Algoritmo IVP

O algoritmo IVP combina os procedimentos heurísticos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Path Relinking* (PR). O VND é responsável por realizar as buscas locais do ILS e o PR é usado como estratégia de intensificação e diversificação da busca. O pseudocódigo deste algoritmo, nomeado IVP, é mostrado no Algoritmo 9.

O Algoritmo 9 recebe como entrada o número de iterações em cada nível de perturbação, *vezesnivel*, e o tempo limite de execução do algoritmo, *tempoExec*. No início da execução do IVP há a inicialização do tempo de execução, *tempoAtual* (linha 1), bem como de três soluções (linha 2), e também ocorre a criação do conjunto de soluções elite (linha 3). Em seguida, na linha 4, é gerada uma solução gulosa pela heurística *Adaptive Shortest Processing Time* (Seção 4.4.1), a partir do número de tarefas, n, do número de máquinas, m, e de uma solução, s. Tal solução passa por processos de buscas locais na linha 5, por meio do módulo VND com ordem estabelecida (Seção 4.4.5). Com o resultado destas buscas locais, a melhor solução conhecida até então, *melhorSol*, é atualizada (linha 6). A seguir, a solução criada é inserida no conjunto elite (linha 7) e o tempo de execução é calculado (linha 9).

O processo iterativo do IVP está situado nas linhas 10 a 42 e termina quando o tempo de execução do algoritmo, *tempoAtual*, for maior que o tempo limite, *tempoExec*. Uma cópia da solução atual é feita na linha 11. O laço seguinte (linhas 14-37) é responsável por controlar o número de iterações em cada nível de perturbação, *vezesnivel*. Durante *vezesnivel* iterações é realizado um número fixo de perturbações. O próximo laço, linhas 17 a 20, executa as perturbações (Seção 4.4.8), sendo que o número delas depende do nível de perturbação. Os níveis de perturbações variam de 1 a 4 e o número de perturbações realizadas em cada nível é dado pelo número do nível de perturbação acrescido de 1; sendo assim, tendo o nível 2 como exemplo, ocorrerão 3 perturbações. Depois de aplicadas as perturbações, a solução obtida é refinada pelo VND com ordem pré-definida (Seção 4.4.5). Em seguida, é verificado se o ótimo local alcançado, em relação a todas vizinhanças adotadas no VND, pode ser inserido no conjunto elite (linhas 21 e 22). Estando o

Algoritmo 9: IVP

Entrada: $f(.), vezesnivel, tempoExec$
Saída: melhorSol

```
1  tempoAtual ← 0;
2  Solucao s, s', melhorSol;
3  elite ← {};
4  s ← geraSolucaoASPT(n,m,s);                    /* Vide Seção 4.4.1 */
5  s ← VND(s);                                    /* Vide Seção 4.4.5 */
6  melhorSol ← s;
7  elite ← elite ∪ {melhorSol};
8  nivel ← 1;
9  Atualize tempoAtual;
10 enquanto tempoAtual ≤ tempoExec faça
11 |  s' ← s;
12 |  vezes ← 0;
13 |  perturbmax ← nivel + 1;
14 |  enquanto vezes < vezesnivel faça
15 |  |  perturb ← 0;
16 |  |  s' ← s;
17 |  |  enquanto perturb < perturbmax faça
18 |  |  |  perturb ++;
19 |  |  |  s' ← perturbacao(s');                 /* Vide Seção 4.4.8 */
20 |  |  fim
21 |  |  s' ← VND(s');                            /* Vide Seção 4.4.5 */
22 |  |  elite ← atualizaElite(s');
23 |  |  pr ← aleatorio(0,1);
24 |  |  se pr ≤ 0.1 e |elite| ≥ 5 então
25 |  |  |  el ← aleatorio(1,5);
26 |  |  |  se f(elite[el]) ≠ f(s') então
27 |  |  |  |  s' ← BkPR(elite [el], s');          /* Vide Seção 4.4.9.1 */
28 |  |  |  fim
29 |  |  fim
30 |  |  se f(s') < f(s) então
31 |  |  |  s ← s';
32 |  |  |  atualizaMelhor(s, melhorSol);
33 |  |  |  elite ← atualizaElite(s);
34 |  |  fim
35 |  |  vezes ++;
36 |  |  Atualize tempoAtual;
37 |  fim
38 |  nivel ++;
39 |  se nivel ≥ 4 então
40 |  |  nivel ← 1;
41 |  fim
42 fim
43 Retorne melhorSol ;
```

conjunto elite completo, aplica-se (linhas 23 a 29), com uma probabilidade de 10%, a estratégia *backward relinking* (Seção 4.4.9.1). Observa-se, no entanto, que tal procedimento somente é aplicado se este ótimo local não for um membro do conjunto elite. Nas linhas 30 a 34 é averiguado se as alterações feitas na solução contribuíram para uma melhor qualidade da mesma. Ao final do processo iterativo a melhor solução obtida é retornada (linha 43).

4.3.2 Algoritmo GIVP

Outro algoritmo proposto utiliza os procedimentos heurísticos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND), *Path Relinking* (PR) e um procedimento de geração da solução inicial tal como na fase de construção do método *Greedy Randomized Adaptive Search Procedure* (GRASP). A partir da solução inicial, o VND executa as buscas locais do ILS e o PR é usado para intensificar e diversificar a busca. O pseudocódigo deste algoritmo, nomeado GIVP, é mostrado no Algoritmo 10.

A única diferença do GIVP (Algoritmo 10) para o IVP (Algoritmo 9) está na linha 4. Ao invés de gerar a solução inicial de forma gulosa, ela é feita tal como na fase de construção GRASP. Sendo assim, é construída uma solução de maneira parcialmente gulosa (Seção 4.4.2).

O objetivo da utilização da fase construtiva do GRASP é aliar características positivas oriundas tanto das construções gulosas quanto das construções aleatórias. As construções gulosas normalmente geram soluções próximas a ótimos locais, o que favorece uma convergência mais rápida a um ótimo local durante o processo iterativo do ILS. Já a construção aleatória tem a vantagem de gerar soluções diferentes a cada execução, o que possibilita uma exploração maior do espaço de busca.

4.3.3 Algoritmo GIVMP

GIVMP é um algoritmo híbrido que combina a fase de construção do procedimento *Greedy Randomized Adaptive Search Procedure* (GRASP), os procedimentos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND), *Path Relinking* (PR) e um modelo de programação inteira mista (PIM) para resolver o Problema do Caixeiro Viajante Assimétrico (PCVA). A ideia do algoritmo é utilizar a fase de construção GRASP para gerar a solução inicial, o VND como responsável por realizar as buscas locais do ILS, o PR usado como estratégia de intensificação e diversificação da busca e o modelo PIM com a intenção de melhorar a melhor solução encontrada, realizando buscas locais na máquina gargalo. O pseudocódigo deste algoritmo é mostrado no Algoritmo 11.

Esta técnica de incluir um modelo PIM que realiza buscas locais é útil para o "polimento" (*polishing*) de soluções, ou seja, fazer ajustes finos que buscas em vizinhanças pequenas não conseguem fazer. Esse é um motivo pelo qual normalmente essa busca vale a pena no final do processo, dado que se inserida no início consumiria muito tempo otimizando soluções pobres lentamente. Na fase inicial a busca em vizinhanças menores tende a ser suficiente.

O Algoritmo 11 difere dos demais apresentados por conta dos módulos utilizados. Na linha 43 há a inclusão do modelo PIM para resolver o PCVA (Seção 4.4.10).

Algoritmo 10: GIVP

Entrada: $f(.), vezesnivel, tempoExec$
Saída: melhorSol

```
1  tempoAtual ← 0;
2  Solucao s, s', melhorSol;
3  elite ← {};
4  s ←geraSolucaoParcialmenteGulosa(0.2);      /* Vide Seção 4.4.2 */
5  s ← VND(s);                                  /* Vide Seção 4.4.5 */
6  melhorSol ← s;
7  elite ← elite ∪ {melhorSol};
8  nivel ← 1;
9  Atualize tempoAtual;
10 enquanto tempoAtual ≤ tempoExec faça
11 |  s' ← s;
12 |  vezes ← 0;
13 |  perturbmax ← nivel + 1;
14 |  enquanto vezes < vezesnivel faça
15 |  |  perturb ← 0;
16 |  |  s' ← s;
17 |  |  enquanto perturb < perturbmax faça
18 |  |  |  perturb ++;
19 |  |  |  s' ← perturbacao(s');              /* Vide Seção 4.4.8 */
20 |  |  fim
21 |  |  s' ← VND(s');                         /* Vide Seção 4.4.5 */
22 |  |  elite ← atualizaElite(s');
23 |  |  pr ← aleatorio(0,1);
24 |  |  se pr ≤ 0.1 e |elite| ≥ 5 então
25 |  |  |  el ← aleatorio(1,5);
26 |  |  |  se f(elite[el]) ≠ f(s') então
27 |  |  |  |  s' ← BkPR(elite [el], s');     /* Vide Seção 4.4.9.1 */
28 |  |  |  fim
29 |  |  fim
30 |  |  se f(s') < f(s) então
31 |  |  |  s ← s';
32 |  |  |  atualizaMelhor(s, melhorSol);
33 |  |  |  elite ← atualizaElite(s);
34 |  |  fim
35 |  |  vezes ++;
36 |  |  Atualize tempoAtual;
37 |  fim
38 |  nivel ++;
39 |  se nivel ≥ 4 então
40 |  |  nivel ← 1;
41 |  fim
42 fim
43 Retorne melhorSol ;
```

Algoritmo 11: GIVMP

Entrada: $vezesnivel$, $tempoExec$, $f(.)$
Saída: melhorSol

```
1  tempoAtual ← 0;
2  Solucao s, s', melhorSol;
3  elite ← {};
4  s ←geraSolucaoParcialmenteGulosa(0.2);      /* Vide Seção 4.4.2 */
5  s ←RVND(s) ;                                /* Vide Seção 4.4.6 */
6  melhorSol ← s;
7  elite ← elite ∪ {melhorSol};
8  nivel ← 1;
9  Atualize tempoAtual;
10 enquanto tempoAtual ≤ tempoExec * 0.95 faça
11 |  s' ← s;
12 |  vezes ← 0;
13 |  perturbmax ← nivel + 1;
14 |  enquanto vezes < vezesnivel faça
15 |  |  perturb ← 0;
16 |  |  s' ← s;
17 |  |  enquanto perturb < perturbmax faça
18 |  |  |  perturb ++;
19 |  |  |  s'←perturbacao(s');              /* Vide Seção 4.4.8 */
20 |  |  fim
21 |  |  s'←RVND(s');                        /* Vide Seção 4.4.6 */
22 |  |  elite ← atualizaElite(s');
23 |  |  pr ← aleatorio(0,1);
24 |  |  se pr ≤ 0.1 e |elite| ≥ 5 então
25 |  |  |  el ← aleatorio(1,5);
26 |  |  |  se f(elite[el]) ≠ f(s') então
27 |  |  |  |  s' ← MxPR(elite [el], s');    /* Vide Seção 4.4.9.2 */
28 |  |  |  fim
29 |  |  fim
30 |  |  se f(s') < f(s) então
31 |  |  |  s ← s';
32 |  |  |  atualizaMelhor(s,melhorSol);
33 |  |  |  elite ← atualizaElite(s);
34 |  |  fim
35 |  |  vezes ++;
36 |  |  Atualize tempoAtual;
37 |  fim
38 |  nivel ++;
39 |  se nivel ≥ 4 então
40 |  |  nivel ← 1;
41 |  fim
42 fim
43 melhorSol ← buscaLocalPIM(melhorSol, tempoExec * 0.05);  /* Vide Seção
   4.4.10 */
44 Retorne melhorSol;
```

Também há uma diferença na linha 27, que representa a troca da estratégia *backward relinking* (Seção 4.4.9.1) pela estratégia *mixed relinking* (Seção 4.4.9.2). Outro módulo que se difere, nas linhas 5 e 21, é o módulo RVND, que é o VND com ordens aleatórias de exploração das estruturas de vizinhanças (Seção 4.4.6).

Uma característica do GIVMP é que seu processo iterativo termina quando o tempo de execução é maior que 95% de *tempoExec*.

Ao final do processo iterativo (linha 43), é aplicado o modelo PIM na melhor solução encontrada (*melhorSol*), restringindo seu tempo de execução em 5% do *tempoExec*.

4.4 Módulos Implementados

Nesta seção, estão descritos os módulos utilizados pelos algoritmos descritos anteriormente. Os módulos de construção da solução inicial estão descritos nas seções 4.4.1 (*Adaptive Shortest Processing Time*) e 4.4.2 (Construção Parcialmente Gulosa). Em seguida, são relacionadas as estruturas de vizinhanças abordadas (Seção 4.4.3), bem como as buscas locais que utilizam tais estruturas (Seção 4.4.4). Duas variações do VND são apresentadas a seguir, uma que define uma ordem de processamento das buscas locais (Seção 4.4.5) e outra cuja ordem de processamento das buscas é aleatória (Seção 4.4.6). Uma avaliação eficiente da função objetivo é mostrada na Seção 4.4.7. Na Seção 4.4.8, as perturbações são descritas. Logo mais, na Seção 4.4.9, está a descrição de como o PR foi aplicado ao UPMSPST. Finalmente, na Seção 4.4.10, é mostrado um modelo de programação inteira mista para resolver o PCVA.

4.4.1 *Adaptive Shortest Processing Time*

A heurística *Adaptive Shortest Processing Time* (ASPT), representada pelo Algoritmo 12, é uma extensão da heurística *Shortest Processing Time* – SPT (Baker, 1974). Ela tem como entrada o número de tarefas n e o número de máquinas m e como saída uma solução, s.

Primeiramente, cria-se um conjunto $N = \{1, .., n\}$ contendo todas as tarefas e um conjunto $M = \{1, .., m\}$ contendo todas as máquinas. A partir desse conjunto N as tarefas são classificadas de acordo com uma função g_k. Esta função é responsável por obter o tempo de conclusão de uma máquina k. Dada uma lista de tarefas candidatas (LC), é avaliada a inserção de cada uma delas em todas as máquinas por meio da função g_k. Procura-se obter em qual máquina a tarefa candidata produzirá o menor tempo de conclusão, o $g_{\min}$. Tal tarefa será sempre inserida na última posição da máquina respectiva.

Se a máquina com o menor tempo de conclusão ainda não tem nenhuma tarefa alocada, seu novo tempo de conclusão será a soma do tempo de processamento da tarefa a ser inserida com o tempo de preparação inicial para tal tarefa. Se essa máquina possui alguma tarefa, seu novo tempo de conclusão será o tempo de conclusão anterior somado ao tempo de processamento da tarefa a ser inserida e ao tempo de preparação da última tarefa presente na máquina. Esse processo de alocação termina quando todas as tarefas forem designadas à alguma máquina,

produzindo então uma solução, s, viável. Esta solução é retornada pela heurística.

Em resumo, $g_{\min} = \min\{g_k(j), \forall j \in N, \forall k \in M\}$, sendo $g_k(j) = p_{jk} + S_{ijk} + T_k$, j a tarefa candidata, i a última tarefa alocada à máquina k e T_k o tempo de conclusão da máquina k.

O algoritmo é dito adaptativo porque a escolha de uma tarefa a ser inserida depende da alocação pré-existente.

4.4.2 Construção Parcialmente Gulosa

A construção parcialmente gulosa é feita de acordo com a fase de construção da metaheurística GRASP, usando como guia a função de avaliação g_k da heurística ASPT (Seção 4.4.1). O Algoritmo 13 apresenta o funcionamento desta construção e recebe como parâmetros α, M e N.

A cada iteração é construída uma Lista Restrita de Candidatos (LRC) das tarefas melhor classificadas da Lista de Tarefas Candidatas (LC), isto é, da lista das tarefas ainda não alocadas. As tarefas j de LC são classificadas de acordo com a função guia g_k. Integram a LRC as tarefas candidatas distintas tal que o tempo de conclusão seja menor ou igual ao valor da soma $g_{\min} + \alpha(g_{\max} - g_{\min})$, sendo $g_{\min}$ o menor tempo de conclusão produzido pelas tarefas de LC, $g_{\max}$ o maior tempo de conclusão produzido pelas tarefas de LC e α o parâmetro GRASP responsável por controlar o quão gulosa será a solução. A seguir, uma tarefa j é escolhida aleatoriamente de LRC e alocada à máquina respectiva. O processo é repetido até que todas as tarefas sejam alocadas.

4.4.3 Estruturas de Vizinhanças

São utilizadas três estruturas de vizinhanças, tendo como base movimentos de trocas e inserções de tarefas.

- A primeira estrutura de vizinhança (NMI) é investigada por meio de aplicações de movimentos de múltipla inserção, os quais são caracterizados por retirar uma tarefa de uma máquina e inseri-la em uma posição de outra máquina.
- A segunda estrutura de vizinhança (NTD) é explorada por meio de movimentos de trocas de tarefas entre máquinas diferentes.
- A terceira e última estrutura de vizinhança (NTM) é analisada a partir de movimentos de trocas de tarefas pertencentes a uma mesma máquina.

4.4.4 Buscas Locais

São descritas, a seguir, as buscas locais implementadas com base nas estruturas de vizinhanças descritas. A primeira busca local (Seção 4.4.4.1) é realizada a partir de movimentos de múltipla inserção. A segunda (Seção 4.4.4.2) é aplicada por meio de movimentos de trocas entre máquinas diferentes. A terceira (Seção 4.4.4.3) e a quarta (Seção 4.4.4.4) são executadas através de movimentos de trocas na mesma máquina, se diferenciando na forma de seleção da próxima solução. Enquanto a terceira busca utiliza a estratégia *First Improvement*, a quarta utiliza *Best Improvement*.

Algoritmo 12: geraSolucaoASPT

Entrada: n, m
Saída: s

```
1  s ← {};
2  N ← {1, .., n};
3  M ← {1, .., m};
4  para cada tarefa ∈ N faça
5      menorTempo ← ∞;
6      maq ← ∞;
7      job ← ∞;
       // para cada tarefa j não alocada
8      para cada j ∈ N faça
9          para cada k ∈ M faça
10             se existe tarefa alguma tarefa alocada na máquina k então
11                 Seja i a última tarefa alocada na máquina k;
12                 Seja T_k o tempo de conclusão da máquina k;
                   // aplicação da função g
13                 g_k(j) = p_jk + S_ijk + T_k;
14                 se g_k(j) ≤ menorTempo então
15                     menorTempo ← g_k(j);
16                     maq ← k;
17                     job ← j;
18                 fim
19             fim
20             senão
                   // aplicação da função g
21                 g_k(j) = p_jk + S_0jk;
22                 se g_k(j) ≤ menorTempo então
23                     menorTempo ← g_k(j);
24                     maq ← k;
25                     job ← j;
26                 fim
27             fim
28             Inserir a tarefa job na máquina maq;
29             N ← N \ {job};
30         fim
31     fim
32 fim
33 Retorne s;
```

Algoritmo 13: geraSolucaoParcialmenteGulosa

Entrada: α, M, N
Saída: s

```
Inicialize a lista de tarefas candidatas LC;
enquanto (|LC| > 0) faça
    Seja g_min o menor tempo de conclusão das tarefas j ∈ LC em todas as
    máquinas k ∈ M e j_1 a tarefa associada a g_min;
    Seja g_max o maior tempo de conclusão das tarefas j ∈ LC | j ≠ j_1 em
    todas as máquinas k ∈ M;
    Seja LRC o conjunto das tarefas candidatas distintas tal que o tempo de
    conclusão seja menor ou igual ao valor da soma g_min + α(g_max − g_min);
    Aleatoriamente, selecione uma tarefa j ∈ LRC e seja k_1 a máquina
    associada;
    Insira em s a tarefa j na máquina k_1;
    Atualize a lista de tarefas candidatas LC;
fim
Retorne s;
```

4.4.4.1 buscaLocalMI

A busca local *buscaLocalMI* é realizada com base na estrutura de vizinhança NMI e utiliza movimentos de múltipla inserção com a estratégia *First Improvement*. O Algoritmo 14 mostra o funcionamento desta busca, recebendo uma solução s.

Nesta busca, cada tarefa de cada máquina é inserida em todas as posições de todas as máquinas. As remoções das tarefas são feitas começando nas máquinas com maiores tempos de conclusão e indo até as máquinas com menores tempos de conclusão. Já as inserções são feitas a partir das máquinas de menores tempos de conclusão até as máquinas de maiores tempos de conclusão.

O movimento é aceito se os tempos de conclusão das máquinas envolvidas são reduzidos. Caso o tempo de conclusão de uma máquina é reduzido e o tempo de conclusão da outra máquina é acrescido, o movimento pode ser aceito se o valor de tempo reduzido for maior que o valor de tempo aumentado. Destaca-se que, mesmo na ausência de melhoria no valor do *makespan*, o movimento pode ser aceito. Ao ocorrer a aceitação de um movimento, a busca é reiniciada e só termina quando se encontra um ótimo local, ou seja, quando não existir nenhum movimento de aceitação para a vizinhança de múltipla inserção.

4.4.4.2 buscaLocalIM

A busca local, nomeada *buscaLocalIM* e representada pelo Algoritmo 15, tem como base a estrutura de vizinhança NTD e realiza movimentos de trocas entre máquinas diferentes. O parâmetro do algoritmo é uma solução s para o UPMSPST.

Para cada par de máquinas são realizadas todas as trocas possíveis de tarefas entre elas. As trocas são feitas das máquinas que possuem maiores tempos de conclusão para aquelas de menores tempos de conclusão. Os critérios de aceitação nessa busca são os mesmos aplicados na primeira busca. Um movimento é aceito se

Algoritmo 14: buscaLocalMI

```
Entrada: s
Saída: s
1  M ← {1, .., m};
2  Seja K1 o conjunto M ordenado decrescentemente pelos tempos de conclusão;
3  Seja K2 o conjunto M ordenado crescentemente pelos tempos de conclusão;
4  para cada máquina k1 ∈ K1 faça
5      para cada tarefa j ∈ k1 faça
6          para cada máquina k2 ∈ K2 faça
7              para cada tarefa i ∈ k2 faça
8                  Seja s' o resultado da realocação de j imediatamente antes de i;
9                  se critério de aceitação for atendido então
10                     s ← s';
11                     Reinicie a busca;
12                 fim
13             fim
14         fim
15     fim
16 fim
17 Retorne s;
```

após a troca de tarefas entre as duas máquinas envolvidas se houver a redução do tempo de conclusão de pelo menos uma máquina e a diferença entre os novos tempos de conclusão for menor que a diferença antes da troca. Assim que um movimento é aceito, a busca é interrompida.

4.4.4.3 buscaLocalEMFirst

A busca local *buscaLocalEMFirst* (Algoritmo 16) é baseada na estrutura de vizinhança NTM e aplica movimentos de trocas na mesma máquina utilizando a estratégia *First Improvement*. Esta busca é realizada a partir de uma solução s, recebida como parâmetro do algoritmo.

Para cada máquina são feitas todas as trocas possíveis entre suas tarefas. A ordem de escolha das máquinas é da máquina de maior valor de tempo de conclusão até a máquina que possui menor valor de tempo de conclusão. O movimento é aceito se o tempo de conclusão da máquina é reduzido. Se houver melhora, a busca é reiniciada e só termina quando for encontrado um ótimo local em relação ao movimento de trocas na mesma máquina.

4.4.4.4 buscaLocalEMBest

A busca local, de nome *buscaLocalEMBest*, também é baseada na estrutura de vizinhança NTM e aplica movimentos de trocas na mesma máquina utilizando a estratégia *Best Improvement*. O Algoritmo 17 mostra como a busca é executada partindo de uma solução s.

São feitas todas as trocas possíveis entre as tarefas de uma máquina. As máquinas

Algoritmo 15: buscaLocalIM

Entrada: s
Saída: s

```
1  M ← {1, .., m};
2  Seja K1 o conjunto M ordenado decrescentemente pelos tempos de conclusão;
3  Seja K2 o conjunto M ordenado crescentemente pelos tempos de conclusão;
4  para cada máquina k1 ∈ K1 faça
5      para cada tarefa j ∈ k1 faça
6          para cada máquina k2 ∈ K2 faça
7              para cada tarefa i ∈ k2 faça
8                  Seja s' o resultado da troca de j com i;
9                  se critério de aceitação for atendido então
10                     s ← s';
11                     Pare a busca;
12                 fim
13             fim
14         fim
15     fim
16 fim
17 Retorne s;
```

Algoritmo 16: buscaLocalEMFirst

Entrada: s, $f(.)$
Saída: Solução s refinada

```
1  M ← {1, .., m};
2  Seja K1 o conjunto M ordenado decrescentemente pelos tempos de conclusão;
3  para cada máquina k1 ∈ K1 faça
4      para cada tarefa j ∈ k1 faça
5          para cada tarefa i ∈ k1 faça
6              se i ≠ j então
7                  Seja s' o resultado da troca de j com i;
8                  se f(s') < f(s) então
9                      s ← s';
10                     Reinicie a busca;
11                 fim
12             fim
13         fim
14     fim
15 fim
16 Retorne s;
```

são escolhidas a partir da máquina representante do *makespan* até a máquina que possui menor valor de tempo de conclusão. O movimento é aceito se o tempo de conclusão da máquina envolvida for reduzido. Se ocorrer melhora nesta máquina, a busca é reiniciada e só termina quando não há nenhum vizinho que seja melhor que a melhor solução encontrada, ou seja, quando estiver em um ótimo local com relação a essa vizinhança.

Algoritmo 17: buscaLocalEMBest

```
   Entrada: s, f(.)
   Saída: Solução s refinada
1  M ← {1, .., m};
2  Seja K_1 o conjunto M ordenado decrescentemente pelos tempos de conclusão;
3  melhorou ← TRUE;
4  para cada máquina k_1 ∈ K_1 faça
5      enquanto melhorou faça
6          melhorou = FALSE;
7          para cada tarefa j ∈ k_1 faça
8              para cada tarefa i ∈ k_1 faça
9                  se i ≠ j então
10                     Seja s' a solução resultante da melhor troca entre as tarefas
                       i e j;
11                 fim
12                 se f(s') < f(s) então
13                     s ← s';
14                     melhorou ← TRUE;
15                 fim
16             fim
17         fim
18     fim
19 fim
20 Retorne s;
```

4.4.5 *Variable Neighborhood Descent* com Ordem Estabelecida

O módulo *Variable Neighborhood Descent* com ordem estabelecida (*VND*), como o próprio nome induz, estabelece uma ordem pré-definida de aplicação das buscas locais. No Algoritmo 18 estão reproduzidos os passos deste módulo, tendo como parâmetro uma solução s.

Neste módulo, a ordem de aplicação das buscas locais é: *buscaLocalMI*, *buscaLocalIM* e *buscaLocalEMFirst*. Existe uma razão por trás desta ordem. Inicialmente, utilizando movimentos de múltipla inserção, tenta-se definir em qual máquina a tarefa será processada. A seguir, com movimentos de troca em máquinas diferentes, tenta-se consertar as tarefas que ainda permanecem nas máquinas "erradas". Finalmente, trocando as tarefas nas mesmas máquinas, assume-se que as tarefas estão

alocadas nas máquinas "certas" e tenta-se, então, otimizar o tempo de conclusão de cada máquina.

O critério de aceitação da nova solução s' depende se esta solução possui um valor de *makespan* menor que o da solução corrente s. Se esse valor de *makespan* for menor que o valor de *makespan* da solução anterior, s, então a solução corrente passa a ser s' e a busca é reiniciada na primeira estrutura de vizinhança. Caso isso não ocorra, a busca continua na próxima estrutura de vizinhança, até que todas as estruturas sejam analisadas.

Algoritmo 18: VND

Entrada: s, $f(.)$
Saída: s

```
1  k ← 1;
2  enquanto (k ≤ 3) faça
3  |  se k = 1 então
4  |  |  s' ← buscaLocalMI(s);
5  |  fim
6  |  se k = 2 então
7  |  |  s' ← buscaLocalIM(s);
8  |  fim
9  |  se k = 3 então
10 |  |  s' ← buscaLocalEMFirst(s);
11 |  fim
12 |  se f(s') < f(s) então
13 |  |  s ← s';
14 |  |  k ← 1;
15 |  fim
16 |  senão
17 |  |  k++;
18 |  fim
19 fim
20 Retorne s;
```

4.4.6 *Variable Neighborhood Descent* com Ordem Aleatória

No módulo *Variable Neighborhood Descent* com ordem aleatória (*Random Neighborhood Variable Descent* – RVND) não existe uma pré-definição da ordem de aplicação das buscas locais. O Algoritmo 19 representa o funcionamento de tal módulo a partir de uma solução s para o UPMSPST.

As buscas locais *buscaLocalMI*, *buscaLocalIM* e *buscaLocalEMBest* são utilizadas por este módulo. Entretanto, não é estabelecida uma ordem de processamento fixa dessas buscas locais, ou seja, a cada chamada do método VND, uma ordem de processamento dessas buscas é definida aleatoriamente. O critério de aceitação da nova solução s' depende se ocorreu a diminuição do *makespan*. Caso isso seja verdade, então a solução corrente s passa a ser s' e a busca é reiniciada com a primeira

estrutura de vizinhança; caso contrário, a busca continua na próxima estrutura de vizinhança e encerra quando todas as vizinhanças forem analisadas.

Algoritmo 19: RVND

Entrada: s, $f(.)$
Saída: s

```
1  v ← {1, 2, 3};
2  Embaralhe v;
3  k ← 1;
4  enquanto (k ≤ 3) faça
5      se k = v[1] então
6          s' ← buscaLocalMI(s);
7      fim
8      se k = v[2] então
9          s' ← buscaLocalEMBest(s);
10     fim
11     se k = v[3] então
12         s' ← buscaLocalIM(s);
13     fim
14     se f(s') < f(s) então
15         s ← s';
16         k ← 1;
17     fim
18     senão
19         k++;
20     fim
21 fim
22 Retorne s;
```

4.4.7 Avaliação Eficiente da Função Objetivo

Avaliar uma solução totalmente a cada movimento de inserção ou troca é muito custoso computacionalmente. Com a intenção de tornar as buscas locais mais eficientes, é utilizado um procedimento que evita essa situação, baseado apenas na avaliação das máquinas que foram modificadas. Com isso, bastam algumas somas e diferenças para se obter o novo tempo de conclusão de cada máquina.

A Figura 4.2 ilustra o procedimento de cálculo da função de avaliação, a partir de um movimento de inserção. Essa figura foi desenvolvida com base na Figura 4.1 e nos dados das tabelas 2.1, 2.2 e 2.3. Percebe-se a retirada da tarefa 6 da máquina M2 e a sua inserção após a tarefa 7 da máquina M1.

O procedimento de avaliação calcula o novo tempo de conclusão da máquina M2 subtraindo, deste, o tempo de processamento da tarefa 6, p_{62}, e também subtraindo os tempos de preparação envolvidos, S_{462} e S_{632}. Além disso, deve ser somado ao tempo de conclusão da máquina M2 o tempo de preparação das novas tarefas adjacentes (4 e 3), S_{432}. O novo tempo de conclusão da máquina M2 é calculado como $M2 = 130 - 23 - 5 - 5 + 8 = 105$.

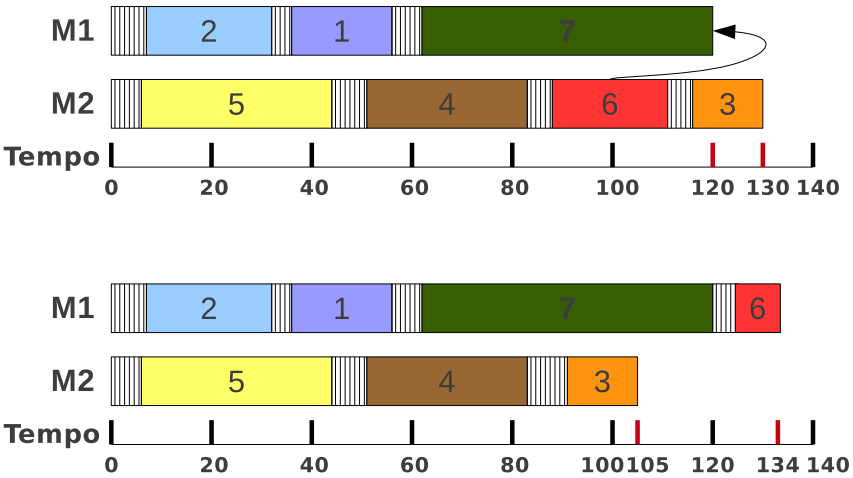

Figura 4.2: Avaliação no movimento de inserção

Já na máquina M1, são somados ao seu tempo de conclusão, o tempo de processamento da tarefa 6 nesta máquina, p_{61}, e o tempo de preparação das novas tarefas adjacentes (7 e 6) S_{761}. Como a tarefa 7 é a última a ser processada, nenhum outro tempo de preparação é necessário, assim, não é preciso subtrair nada. O novo tempo de conclusão da máquina M1 é calculado pela equação $M1 = 120 + 9 + 5 = 134$.

Foi dado um exemplo da aplicação desse procedimento para avaliar um movimento de inserção. Ao tratar movimentos de trocas, a aplicação desse procedimento se torna trivial.

Todas as buscas locais descritas na Seção 4.4.4 utilizam este procedimento.

4.4.8 Perturbações

As perturbações consistem em aplicar movimentos de inserção a um ótimo local. Cada perturbação se caracteriza por retirar uma tarefa de uma máquina e inseri-la em outra máquina. No Algoritmo 20 está o pseudocódigo da perturbação de uma solução s, parâmetro do algoritmo.

A escolha das máquinas e da tarefa é feita aleatoriamente. Ao inserir a tarefa em outra máquina, procura-se a melhor posição para ela ser inserida, ou seja, uma posição em que a máquina tiver o menor tempo de conclusão. Desta forma, subpartes do problema são otimizadas a cada perturbação.

Algoritmo 20: perturbacao

Entrada: s
Saída: s

1 Aleatoriamente selecione uma máquina m_1 e uma máquina m_2 de s;
2 Aleatoriamente selecione uma tarefa j de m_1;
3 Remova j de m_1;
4 Insira j na melhor posição na máquina m_2;
5 **Retorne** s;

A quantidade de modificações em uma solução é controlada por um "nível" de perturbação, de forma que um determinado nível n de perturbação consiste na aplicação de $n+1$ movimentos de inserção. O nível máximo permitido para as perturbações é 3; assim, ocorrerão 4 perturbações no máximo. O objetivo de se aumentar o nível de perturbação é diversificar a busca e procurar por uma solução de melhora em uma região gradativamente mais "distante" daquele ótimo local. O nível de perturbação somente é aumentado após geradas *vezesnivel* soluções perturbadas sem que haja melhoria da solução corrente. Por outro lado, sempre que uma melhor solução é encontrada, a perturbação volta ao seu nível mais baixo.

A avaliação eficiente da função objetivo (Seção 4.4.7) utilizada nas buscas locais também é utilizada ao avaliar as perturbações.

4.4.9 *Path Relinking*

A estratégia *Path Relinking* (PR) necessita de um conjunto de soluções elite, isto é um conjunto com soluções de alta qualidade. Este conjunto possui duas regras para que uma solução faça parte dele. Uma solução é incluída no conjunto elite se ela tiver um menor valor de *makespan* do que o valor da melhor solução já presente no mesmo. Também é incluída no conjunto elite a solução que for melhor do que a pior solução do conjunto, mas que satisfaça a um nível mínimo de diversidade das outras soluções elite. Nos algoritmos implementados o tamanho do conjunto elite está limitado a 5 soluções e o nível mínimo de diversidade entre as soluções é de 10%.

O índice de diversidade entre duas soluções para o UPMSPST é definido como o percentual de tarefas diferentes nas mesmas posições das máquinas. Para encontrar esse indicador, uma comparação entre as tarefas a cada posição da solução é realizada. O somatório das posições que apresentarem tarefas diferentes é, então, dividido pelo total de posições da solução. O resultado desta divisão corresponde ao índice de diversidade entre as soluções avaliadas.

Para ilustrar o cálculo do índice de diversidade, seja a Figura 4.3 na qual estão representadas duas soluções, s e s'. O número de tarefas diferentes nas mesmas posições é 4, logo o índice de diversidade é dada por 4 dividido por 7, número total de posições. Desta maneira, o índice de diversidade entre s e s' é, em porcentagem, 57%.

De posse do conjunto elite, pode-se então construir os caminhos entre as soluções de alta qualidade, partindo de uma solução base e indo em direção a uma solução guia. Nos algoritmos desenvolvidos a solução base é escolhida, aleatoriamente, como uma das soluções presentes no conjunto elite e a solução guia é a solução resultante da aplicação do VND, isto é, um ótimo local.

Tendo isto em vista, foram utilizadas duas estratégias de aplicação do PR: a estratégia *backward relinking* (Seção 4.4.9.1) e a estratégia *mixed relinking* (Seção 4.4.9.2).

4.4.9.1 *Backward Path Relinking*

A ideia do *Backward Path Relinking* (BkPR) é construir um caminho da melhor solução (s_{base}) para a pior solução (s_{guia}). O Algoritmo 21 mostra como isso é feito

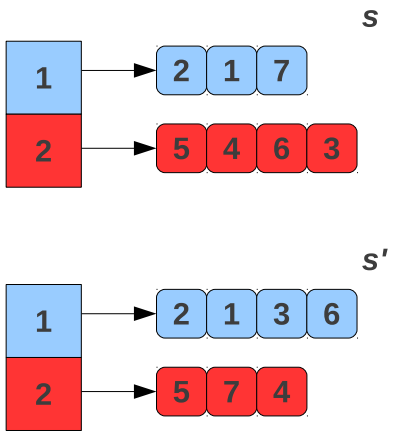

Figura 4.3: Exemplo do cálculo da diversidade entre duas soluções

para o UPMSPST, possuindo os parâmetros s_{base} e s_{guia}.

Algoritmo 21: BkPR

Entrada: s_{elite}, s_{vnd}, $f(.)$
Saída: melhor

```
1  melhor ← arg min(f(s_elite), f(s_vnd)) ;
2  s_base ← melhor;
3  s_guia ← arg max(f(s_elite), f(s_vnd));
4  para cada máquina faça
5      para cada tarefa j de s_guia faça
6          Seja pos1 a posição de j em s_guia;
7          Seja pos2 a posição de j em s_base;
8          Remova j de pos2;
9          Insira j na posição pos1 em s_base;
10         s' ← s_base;
11         s' ← buscaLocalMI(s');
12         se f(s') < f(melhor) então
13             melhor ← s';
14         fim
15     fim
16 fim
17 Retorne melhor;
```

O atributo considerado no Algoritmo 21 é a posição de uma tarefa em s_{guia}. A cada iteração do *BkPR*, uma tarefa de s_{base} é realocada na mesma posição em que esta se encontra em s_{guia}. Feito isso, é realizada uma busca local com movimentos de múltipla inserção nesta nova solução. Este procedimento é repetido até que a s_{base} seja igual à s_{guia}. No final é retornada a melhor solução encontrada durante o processo.

4.4.9.2 *Mixed Path Relinking*

A estratégia *Mixed Path Relinking* (MxPR) procura explorar dois caminhos simultaneamente. O primeiro parte da solução base (s_{base}) como sendo a pior solução e vai em direção à solução guia (s_{guia}), considerada a melhor solução. O segundo parte da solução base (s_{base}) como sendo a melhor solução e caminha até a pior solução, solução guia (s_{guia}). O fim desta estratégia acontece quando esses caminhos se encontram em uma solução intermediária equidistante de ambas soluções (s_{base} e s_{guia}). O Algoritmo 22 mostra a aplicação desta estratégia para o UPMSPST, recebendo como entrada uma solução base s_{base} e uma solução guia s_{guia}.

Algoritmo 22: MxPR

Entrada: s_{base}, s_{guia}
Saída: s'
1 Seja Δ o conjunto de todas as posições das tarefas de s_{base} que se diferem de s_{guia};
2 $s' \leftarrow \arg\min\{somaMaq(s_{base}), somaMaq(s_{guia})\}$;
3 $x \leftarrow s_{base}$;
4 **enquanto** $(|\Delta| > 1)$ **faça**
5 $\quad l^* \leftarrow \arg\min\{somaMaq(x \oplus l) : l \in \Delta\}$;
6 $\quad \Delta \leftarrow \Delta \setminus \{l^*\}$;
7 $\quad x \leftarrow x \oplus l^*$;
8 $\quad AtualizaMelhor(x, s')$; `/* s' é a melhor solução encontrada */`
9 $\quad s_{base} \leftarrow s_{guia}$;
10 $\quad s_{guia} \leftarrow x$;
11 $\quad$ Compute Δ com todas as posições das tarefas de s_{base} que se diferem de s_{guia};
12 $\quad s' \leftarrow buscaLocalMI(s')$;
13 **fim**
14 **Retorne** s';

No Algoritmo 22 o atributo considerado também é a posição de uma tarefa. Inicialmente gera-se o conjunto Δ que contém as posições das tarefas de s_{base} que se diferem das posições das tarefas de s_{guia}. Até que todas as tarefas de s_{guia} estejam nas mesmas posições em s_{base}, deve-se colocar em s_{base} as tarefas nas posições contidas em Δ. Após uma tarefa ser inserida em s_{base}, deve-se calcular a soma dos tempos de conclusão, $somaMaq$, para esta nova solução. O movimento que obtiver o menor resultado de $somaMaq$ será aplicado em s_{base} e removido de Δ. Em seguida, a melhor solução encontrada s' é atualizada.

As linhas 9, 10 e 11 caracterizam a estratégia *mixed relinking*. Nas linhas 9 e 10 as funções de s_{guia} e s_{base} são trocadas. Após esta troca, na linha 11 é calculado um novo Δ contendo as posições das tarefas de s_{base} que se diferem das posições das tarefas de s_{guia}. Na linha 12 é aplicada a busca local, *buscaLocalMI* (Seção 14), em s'.

Finalmente, após $|\Delta| = 0$, o MxPR termina e retorna a melhor solução encontrada ao longo de sua execução.

4.4.10 Modelo de Programação Inteira Mista

O modelo de Programação Inteira Mista utilizado no GIVMP é baseado na formulação de Sarin et al. (2005), que, por sua vez, foi desenvolvida para resolver o Problema do Caixeiro Viajante Assimétrico (PCVA). Tanto o PCVA (Seção 4.4.10.1) quanto a formulação (Seção 4.4.10.2) e uma busca local que a utiliza (Seção 4.4.10.3) serão explicados a seguir.

4.4.10.1 Problema do Caixeiro Viajante Assimétrico

O Problema do Caixeiro Viajante Assimétrico (PCVA) é caracterizado por conter um conjunto finito de cidades, $N = \{1, .., n\}$, e também por haver um custo de viagem de uma cidade $i \in N$ para uma cidade $j \in N$, c_{ij}. O fato de esse problema ser assimétrico implica que o custo c_{ij} é diferente do custo c_{ji}, para qualquer par $i, j \in N$. O objetivo desse problema é encontrar um circuito com a menor distância possível que visite cada cidade exatamente uma vez.

A escolha do PCVA foi realizada pois o mesmo se equivale ao problema de encontrar a melhor sequência das tarefas em uma máquina (PSM). Tomando as cidades como cada tarefa de uma máquina e o custo de viagem como cada tempo de processamento somado ao tempo de preparação, quando existir, essa equivalência é facilmente percebida. Sendo assim, a resolução do PCVA para uma máquina e suas tarefas retorna o menor tempo de conclusão desta máquina com as respectivas tarefas.

Para ilustrar a equivalência do PCVA com o PSM, toma-se a Figura 4.4. A figura foi criada com base nas tarefas da máquina M1 da Figura 4.1 e nos dados das tabelas 2.1 e 2.2.

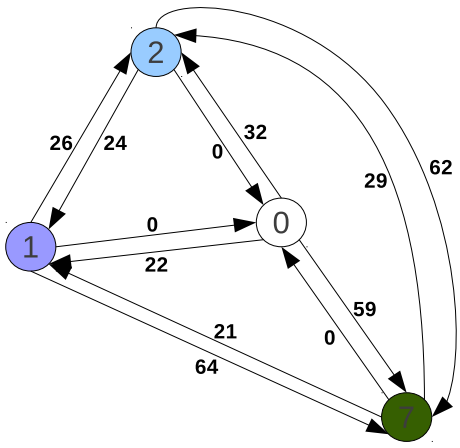

Figura 4.4: Exemplo da equivalência do PCVA com o PSM

Na figura, cada vértice, ao invés de representar uma cidade, representa uma tarefa. Ao realizar a conversão do PSM para o PCVA deve ser adicionada uma tarefa fictícia, representada na figura como a tarefa 0. Um caminho entre dois vértices representa o sequenciamento da tarefa simbolizada pelo vértice de destino (V_d) após a tarefa simbolizada pelo vértice de origem (V_o). O custo desse caminho é calculado pelo tempo de processamento da tarefa do V_d somado ao tempo de

preparação entre a tarefa do V_o e a tarefa do V_d. Por exemplo, o custo de ir da tarefa 2 para a tarefa 1, na máquina 1, é calculado pela soma $p_{11} + s_{211}$. O custo do caminho entre 0 e alguma tarefa é dado pelo tempo de processamento desta tarefa somado ao tempo de preparação inicial para a mesma. O custo de ir de alguma tarefa para a tarefa 0 é igual a 0, pois esse caminho representa o fim do sequenciamento. Desta forma, o circuito [0,2,1,7,0] representa o mesmo sequenciamento da máquina M1, com um custo total de 120.

4.4.10.2 Formulação de Sarin et al. (2005)

A formulação de Sarin et al. (2005), nomeada SSB1, foi escolhida por ser uma formulação que apresenta um tempo computacional relativamente menor em relação a outras da literatura, como mostrado em Öncan et al. (2009).

Tal formulação define as variáveis binárias de decisão $x_{ij} = 1$ se a cidade i precede imediatamente a cidade j em um circuito e $x_{ij} = 0$, caso contrário. Além disso, também são definidas as variáveis binárias de decisão $y_{ij} = 1$ se a cidade i precede, não necessariamente imediatamente, a cidade j em um circuito e $y_{ij} = 0$ caso contrário.

Sendo assim, os autores propuseram o seguinte modelo:

$$min \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \tag{4.1}$$

$$s.a.$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \qquad \forall \quad j = 1, \ldots, n \tag{4.2}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \qquad \forall \quad i = 1, \ldots, n \tag{4.3}$$

$$y_{ij} \geq x_{ij} \qquad \forall \quad i, j = 2, \ldots, n, i \neq j \tag{4.4}$$

$$y_{ij} + y_{ji} = 1 \qquad \forall \quad i, j = 2, \ldots, n, i \neq j \tag{4.5}$$

$$y_{ij} + y_{jk} + y_{ki} \leq 2 \qquad \forall \quad i, j, k = 2, \ldots, n, i \neq j \neq k \tag{4.6}$$

$$y_{j1} = 0 \qquad \forall \quad j = 2, \ldots, n, \tag{4.7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \quad i, j = 1, \ldots, n, i \neq j \tag{4.8}$$

$$y_{ij} \in \{0, 1\} \qquad \forall \quad i, j = 1, \ldots, n, i \neq j \tag{4.9}$$

O objetivo (4.1) é minimizar o custo total da viagem. As restrições (4.2) e (4.3) asseguram que cada vértice é incidente a um arco de saída e um arco de entrada. As restrições (4.4) a (4.6) servem para prevenir a ocorrência de subcircuitos, isto é, impedir que se forme um aglomerado de cidades sem ligação. Estas últimas restrições, (4.4) a (4.6), em conjunto com as outras, (4.2) e (4.3) garantem uma formulação válida para o PCVA. A restrição (4.7) garante que nenhuma cidade será antecessora da cidade 1 que é considerada como origem. As restrições (4.8) e (4.9) definem o domínio das variáveis binárias.

Ao longo do trabalho vários testes foram realizados com o objetivo de acelerar a resolução deste modelo. Dessa forma, a versão que se mostrou mais eficiente foi obtida pela retirada da restrição (4.6) e pela inclusão das restrições de fluxo do modelo de Laporte (1992), onde f_{ij} representa o fluxo da cidade i para a cidade j. As restrições incluídas são mostradas a seguir:

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} f_{ji} - \sum_{\substack{i=1 \\ i \neq j}}^{n} f_{ij} = 1 \qquad \forall \quad j = 2, \ldots, n \tag{4.10}$$

$$f_{ij} \leq (n-1)x_{ij} \qquad \forall \quad i, j = 1, \ldots, n, i \neq j \tag{4.11}$$

$$f_{ij} \geq 0 \qquad \forall \quad i, j = 1, \ldots, n, i \neq j \tag{4.12}$$

4.4.10.3 Busca Local Baseada em um modelo PIM

A busca local *buscaLocalPIM* é baseada na formulação de Sarin et al. (2005) apresentada e está descrita no Algoritmo 23, que recebe a melhor solução encontrada pelo processo iterativo do GIVMP, s, e o tempo limite de sua execução, *tempo*.

Primeiramente, deve-se procurar a máquina gargalo k, ou seja, a máquina cujo tempo de conclusão é o *makespan*. Em seguida, deve-se calcular a matriz de distâncias entre as tarefas para essa máquina. Essa matriz tem o tamanho $(n+1)X(n+1)$, onde n é o número de tarefas alocadas em k. Tal matriz é construída a partir de regras simples. Seja i o número que representa a linha da matriz e j o número que representa a coluna. Se $i = j$ ou $j = 0$ o custo será zero. Se $i = 0$ representa que a tarefa será a primeira a ser alocada, então deve-se tomar o tempo de preparação inicial para essa tarefa somado ao tempo de processamento da mesma. Caso não entre em nenhum caso citado, significa que a tarefa está no meio da sequência e, nesta situação, deve ser calculado o tempo de preparação em relação a tarefa antecessora a ela somado ao seu tempo de processamento.

Por meio da matriz de distâncias pode-se, então, resolver o modelo PIM descrito por meio de um resolvedor matemático, na tentativa de obter uma sequência melhor para a máquina gargalo (linha 4). Para agilizar a busca, o sequenciamento já existente em k é passado como solução inicial para resolvedor matemático. O tempo limite para resolver esse modelo PIM é o *tempo* passado como parâmetro. Se a nova sequência encontrada tiver o tempo de conclusão menor que o da sequência anterior, então esta nova sequência é atribuída à máquina k. A seguir, o *makespan* é atualizado. Se ainda tem tempo para mais uma busca, primeiro é verificado se a máquina gargalo foi alterada; caso isso aconteça é realizada uma nova busca local, *buscaLocalPIM*, nessa máquina. Se a máquina gargalo continua a mesma, são realizadas chamadas iterativas do módulo RVND, aplicado a s, seguido do módulo *MxPR*, aplicado a alguma solução do conjunto elite e a s. Assim que o tempo acabar, é retornada a solução s tendo as sequências de suas máquinas otimizadas.

Algoritmo 23: buscaLocalPIM

Entrada: $s, tempo, f(.)$
Saída: s

Seja k a máquina que representa o *makespan* de s;
Seja T_k o tempo de conclusão da máquina k;
$matrizDist[\,][\,] \leftarrow criaMatrizDistancias(k)$;
$k' \leftarrow SSB1(matrizDist, tempo)$;
Seja $T_{k'}$ o tempo de conclusão da máquina k';
se $T_{k'} < T_k$ **então**
    $k \leftarrow k'$;
fim
Atualize *tempoAtual*;
se *tempoAtual* $<$ *tempo* **então**
    Seja k'' a máquina que representa o novo *makespan* de s;
    se $k'' \neq k$ **então**
        $s \leftarrow buscaLocalPIM(s, tempo)$;
        Atualize *tempoAtual*;
    fim
    senão
        enquanto *tempoAtual* $\leq$ *tempo* **faça**
            $s \leftarrow RVND(s)$; `/* Vide Seção 4.4.6 */`
            $el \leftarrow aleatorio(1, 5)$;
            $s \leftarrow MxPR(elite[el], s)$; `/* Vide Seção 4.4.9.2 */`
            Atualize *tempoAtual*;
        fim
    fim
fim
Retorne s;

Capítulo 5

Resultados Computacionais

Neste Capítulo são apresentados os resultados computacionais obtidos pelos algoritmos propostos. Na Seção 5.1 os problemas-teste utilizados são descritos. Na Seção 5.2 são apresentados e analisados os resultados obtidos pelos três algoritmos propostos. Nas seções 5.3 e 5.4, os resultados alcançados com as instâncias de Vallada e Ruiz (2011) e Rabadi et al. (2006), respectivamente, são mostrados e analisados.

Todos os algoritmos foram desenvolvidos na linguagem C++ e todos os experimentos foram executados em um computador com processador *Intel Core 2 Quad 2,4 GHz* com 4 GB de memória RAM e sistema operacional *Linux*, *Ubuntu 10.10*. Para executar o modelo PIM proposto foi utilizado o otimizador CPLEX versão 12.1, em sua configuração padrão.

5.1 Problemas-teste

Nesta seção são apresentados os problemas-teste utilizados para testar os algoritmos desenvolvidos. Primeiro, na Seção 5.1.1, o conjunto de instâncias criadas em Vallada e Ruiz (2011) é descrito e logo após, na Seção 5.1.2, são apresentados os problemas-teste desenvolvidos em Rabadi et al. (2006).

5.1.1 Instâncias de Vallada e Ruiz (2011)

Vallada e Ruiz (2011) geraram dois conjuntos de instâncias. Seja n o número de tarefas e m o número de máquinas. O primeiro conjunto é composto por instâncias pequenas com combinações de $n \in \{6, 8, 10, 12\}$ e $m \in \{2, 3, 4, 5\}$. O segundo conjunto possui instâncias grandes, envolvendo combinações de $n \in \{50, 100, 150, 200, 250\}$ e $m \in \{10, 15, 20, 25, 30\}$. Para cada combinação de máquinas e tarefas foram geradas 40 instâncias, totalizando 640 instâncias pequenas e 1000 instâncias grandes.

Em ambos os conjuntos, os tempos de processamento são obtidos por meio de uma distribuição uniforme $U[1, 99]$. São gerados 4 subconjuntos através dos tempos de preparação, contendo 10 instâncias cada. Esses 4 subconjuntos são gerados pelas distribuições uniformes: $U[1, 9]$, $U[1, 49]$, $U[1, 99]$ e $U[1, 124]$.

Todas essas instâncias são encontradas em SOA (2011), onde também são encontrados os melhores valores de *makespan* para as mesmas. Estes melhores valores foram encontrados em Vallada e Ruiz (2011) durante os experimentos computacio-

nais com os algoritmos por eles implementados, bem como pelos algoritmos de Kurz e Askin (2001) e Rabadi et al. (2006) que foram reimplementados.

5.1.2 Instâncias de Rabadi et al. (2006)

Rabadi et al. (2006) também geraram 2 conjuntos de instâncias, a partir de n tarefas e m máquinas. Um com instâncias pequenas que combinam $n \in \{6, 7, 8, 9, 10, 11\}$ com $m \in \{2, 4, 6, 8\}$; e o outro com instâncias grandes compostas por combinações de $n \in \{20, 40, 60, 80, 100, 120\}$ e $m \in \{2, 4, 6, 8, 10, 12\}$. O conjunto de instâncias pequenas é constituído de 810 problemas-teste, enquanto o conjunto de instâncias grandes possui 1620 instâncias.

Para cada combinação de tarefas e máquinas existem 3 grupos de 15 instâncias cada. Tais grupos são definidos por um nível de domínio. O primeiro grupo é gerado com os tempos de preparação e de processamento balanceados, ou seja, ambos são extraídos de uma distribuição uniforme $U[50, 100]$. O segundo grupo é definido por tempos de processamento dominantes, isto é, os tempos de processamento são obtidos da distribuição uniforme $U[125, 175]$ e os tempos de preparação extraídos da distribuição uniforme $U[50, 100]$. Já o terceiro grupo é caracterizado por conter tempos de preparação dominantes, ou seja, os tempos de processamento são gerados por uma distribuição uniforme $U[50, 100]$ e os tempos de preparação são gerados a partir de uma distribuição uniforme $U[125, 175]$.

Os autores disponibilizaram todas essas instâncias em SchedulingResearch (2005), onde estão também os melhores resultados obtidos em Rabadi et al. (2006), em Helal et al. (2006) e Arnaout et al. (2010). Destes trabalhos o que apresentou os melhores resultados foi o ACO de Arnaout et al. (2010).

5.2 Algoritmos Desenvolvidos

Nesta seção são mostrados e analisados os resultados encontrados pelos três algoritmos desenvolvidos (IVP, GIVP e GIVMP).

Todos os três algoritmos foram testados utilizando 15 instâncias grandes, sendo 9 problemas-teste disponíveis em SOA (2011) e 6 disponíveis em SchedulingResearch (2005). Cada uma das 9 instâncias disponíveis em SOA (2011) origina de cada combinação de $n \in \{50, 100, 150\}$ e $m \in \{10, 15, 20\}$. Das 6 instâncias disponíveis em SchedulingResearch (2005), cada uma descende de cada combinação de $n \in \{80, 100\}$ e $m \in \{4, 6, 10\}$.

O parâmetro *vezesnivel* utilizado nos algoritmos IVP, GIVP e GIVMP foi igual a 15, obtido através de experimentos empíricos. Também a partir de experimentos empíricos definiu-se o parâmetro $\alpha = 0,2$ de *geraSolucaoParcialmenteGulosa* (presente em GIVP e GIVMP). O critério de parada de todos os algoritmos foi o tempo máximo de duração do processamento *tempoExec*, em milissegundos. Tal tempo é obtido pela Eq. (5.1), em que m representa o número de máquinas, n o número de tarefas e t é um parâmetro que varia entre 10, 30 e 50. Observa-se que esse critério de parada, com estes valores de t, foi o mesmo adotado em Vallada e Ruiz (2011). Para os testes com os três algoritmos desenvolvidos também foram utilizados os mesmos valores para t (10, 30 e 50).

$$tempoExec = n \times (m/2) \times t \;\; ms \tag{5.1}$$

Na métrica utilizada para a comparação dos algoritmos, dada pela Eq. (5.2), o objetivo é verificar a variabilidade das soluções finais produzidas pelos algoritmos. Nesta medida, calcula-se, para cada algoritmo Alg aplicado a um problema-teste i, o desvio percentual relativo RPD_i da solução encontrada $\bar{f}_i^{Alg}$ em relação à melhor solução f_i^* conhecida até então. Todos os três algoritmos propostos foram executados 30 vezes para cada instância. Ao final dessas 30 execuções calcula-se, para cada instância, a média RPD_{avg} dos valores de RPD_i encontrados.

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \tag{5.2}$$

Ao comparar os resultados obtidos utilizando as instâncias disponíveis em SOA (2011), estes resultados são comparados com as melhores soluções obtidas em Vallada e Ruiz (2011) durante a execução dos testes. Em contrapartida, ao realizar a comparação com os problemas-teste disponíveis em SchedulingResearch (2005), os resultados são comparados com um limite inferior proposto por Al-Salem (2004). O limite inferior (LB) é considerado relativamente fraco pelo próprio autor que o propôs e é calculado pelas Eqs. (5.3), (5.4), e (5.5). Os valores desses limites também são encontrados em SchedulingResearch (2005).

$$LB1 = \frac{1}{m}\sum_{j=1}^{n} \min[p_{jk} + s_{ijk}] \qquad \forall i \in N, k \in M \tag{5.3}$$

$$LB2 = \max\{\min[p_{jk} + s_{ijk}]\} \qquad \forall i, j \in N, k \in M \tag{5.4}$$

$$LB = \max\{LB1, LB2\} \tag{5.5}$$

As Tabelas 5.1, 5.2 e 5.3 mostram para cada instância (coluna *Instância*), a média (coluna *Média*), a mediana (coluna *Mediana*), o desvio padrão (coluna *Desv Pad*), o desvio percentual relativo médio (coluna RPD_{avg}, em porcentagem) e o desvio percentual relativo à melhor solução conhecida (coluna RPD_{best}, em porcentagem) dos algoritmos IVP, GIVP e GIVMP, respectivamente. Também são mostrados, nessas Tabelas, esses valores para cada valor do parâmetro t utilizado (colunas 10, 30 e 50). A última linha de cada divisão indica a média total para cada valor apresentado. Os valores negativos de RPD representam que os resultados encontrados superam as melhores soluções de Vallada e Ruiz (2011).

A partir dos resultados apresentados pode-se perceber que o algoritmo GIVMP possui os melhores valores de RPD, não só com relação às médias (RPD_{avg}) obtidas, mas também com relação ao melhor valor (RPD_{best}) obtido em cada instância. Além de ganhar em todos os conjuntos de problemas-teste, ele ainda melhorou a maioria dos melhores resultados de Vallada e Ruiz (2011). O segundo melhor algoritmo foi o IVP, possuindo melhores valores de RPD_{avg} e RPD_{best} que o GIVP. Este, por sua vez, não conseguiu gerar soluções melhores que os demais, talvez por partir de uma solução parcialmente gulosa e precisar de mais tempo para chegar a uma solução de qualidade ou uma técnica mais eficiente de intensificação.

O algoritmo que obteve os menores desvios padrões foi o IVP, porém a diferença entre esses valores é mais perceptível aos compará-los com os desvios padrões do

Tabela 5.1: Resultados obtidos pela aplicação do algoritmo IVP

IVP	**Média**			**Mediana**			**Desv Pad**			RPD_{avg}			RPD_{best}		
Instância	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**
I_50_10_S_1-124_1	115,233	112,9	110,933	115	112	110	3,63	3,37	3,36	1,08%	-0,96%	-2,69%	-4,39%	-6,14%	-8,77%
I_50_15_S_1-124_1	75,8667	74,8333	74,4	75	75	75	2,66	2,15	2,77	1,16%	-0,22%	-0,80%	-4,00%	-5,33%	-6,67%
I_50_20_S_1-124_1	54,3333	52,8333	52,1667	54	53	53	3,07	2,90	2,98	4,49%	1,60%	0,32%	-5,77%	-5,77%	-7,69%
I_100_10_S_1-124_1	230,033	226,4	223,833	230,5	225	224	7,92	7,48	4,68	4,09%	2,44%	1,28%	-2,26%	-1,81%	-2,26%
I_100_15_S_1-124_1	138,733	136,033	135,8	138	136	136,5	4,86	4,16	3,17	-1,61%	-3,52%	-3,69%	-7,80%	-9,22%	-8,51%
I_100_20_S_1-124_1	101,133	97,4667	97,8667	99,5	97,5	97,5	5,04	4,75	5,69	1,13%	-2,53%	-2,13%	-9,00%	-13,00%	-10,00%
I_150_10_S_1-124_1	333,8	330,833	325,167	334	331	324,5	7,43	7,79	6,68	0,54%	-0,35%	-2,06%	-4,22%	-4,52%	-5,12%
I_150_15_S_1-124_1	206,767	202,233	201,2	206,5	202	202	4,02	3,36	5,09	-1,54%	-3,70%	-4,19%	-5,71%	-7,62%	-9,52%
I_150_20_S_1-124_1	137,933	135,333	133,067	138,5	135,5	134	4,87	3,77	5,15	-2,18%	-4,02%	-5,63%	-9,22%	-9,93%	-12,06%
80on4Rp50Rs50_1	2327,23	2316,9	2311,47	2328	2318	2310	13,31	7,91	12,35	6,02%	5,55%	5,31%	4,74%	4,87%	4,28%
80on6Rp50Rs50_1	1539,5	1533,1	1529,97	1539,5	1534	1529	6,51	6,20	7,98	7,66%	7,21%	6,99%	6,85%	6,22%	5,87%
80on10Rp50Rs50_1	908,8	907,433	904,033	908,5	907	904,5	5,60	5,88	3,77	8,23%	8,07%	7,66%	7,06%	6,70%	6,59%
100on4Rp50Rs50_1	2880,8	2874,63	2868,73	2880,5	2874,5	2867,5	8,54	10,40	10,90	5,17%	4,94%	4,73%	4,55%	4,12%	4,08%
100on6Rp50Rs50_1	1883,1	1877,93	1876,3	1882	1878	1876	5,83	4,88	4,79	5,89%	5,60%	5,51%	5,04%	4,99%	4,82%
100on10Rp50Rs50_1	1122,33	1119,57	1116,8	1123	1119	1117	5,76	5,86	4,21	7,82%	7,56%	7,29%	6,54%	6,64%	6,45%
Média Total	**803,71**	**799,90**	**797,45**	**803,50**	**799,83**	**797,37**	**5,94**	**5,39**	**5,57**	**3,20%**	**1,84%**	**1,19%**	**-1,17%**	**-1,99%**	**-2,57%**

Tabela 5.2: Resultados obtidos pela aplicação do algoritmo GIVP

GIVP	**Média**			**Mediana**			**Desv Pad**			RPD_{avg}			RPD_{best}		
Instância	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**
I_50_10_S_1-124_1	117	114,467	113,3	116	114	113	5,64	4,60	3,97	2,63%	0,41%	-0,61%	-4,39%	-6,14%	-5,26%
I_50_15_S_1-124_1	78,5333	75,3667	74,8333	78	75	74,5	3,46	3,47	2,74	4,71%	0,49%	-0,22%	-2,67%	-6,67%	-5,33%
I_50_20_S_1-124_1	53,6333	53,7	52,2	53	53	53	2,59	3,71	2,98	3,14%	3,27%	0,38%	-5,77%	-5,77%	-11,54%
I_100_10_S_1-124_1	237,267	230,867	227,9	235,5	230	229,5	10,33	7,36	8,22	7,36%	4,46%	3,12%	-2,26%	-1,81%	-2,71%
I_100_15_S_1-124_1	144	139,6	140,4	143	140	140	4,98	4,99	6,84	2,13%	-0,99%	-0,43%	-4,96%	-7,80%	-10,64%
I_100_20_S_1-124_1	100,733	98,2333	96,6333	100,5	97,5	96,5	5,41	4,98	3,68	0,73%	-1,77%	-3,37%	-10,00%	-9,00%	-9,00%
I_150_10_S_1-124_1	350,267	336	333,7	348,5	333	334	12,68	12,78	13,00	5,50%	1,20%	0,51%	-1,81%	-5,12%	-5,42%
I_150_15_S_1-124_1	220,833	216,033	213,467	222	215,5	216	9,66	8,96	6,33	5,16%	2,87%	1,65%	-1,90%	-4,76%	-4,29%
I_150_20_S_1-124_1	138,7	134,833	133,233	139	134	135	5,83	4,93	7,53	-1,63%	-4,37%	-5,51%	-9,93%	-9,93%	-17,73%
80on4Rp50Rs50_1	2323,93	2316,13	2312,07	2322	2312,5	2310	13,38	14,89	11,85	5,87%	5,52%	5,33%	4,56%	4,60%	4,56%
80on6Rp50Rs50_1	1547,57	1541,5	1535,8	1546	1541,5	1536,5	9,20	8,47	8,36	8,22%	7,80%	7,40%	7,13%	6,71%	6,36%
80on10Rp50Rs50_1	920,167	914,633	910,367	920	914	911,5	9,40	6,92	6,64	9,58%	8,92%	8,42%	7,54%	7,30%	7,06%
100on4Rp50Rs50_1	2904	2887,33	2878,53	2905,5	2888	2876	15,72	10,55	11,88	6,01%	5,41%	5,08%	4,85%	4,63%	4,44%
100on6Rp50Rs50_1	1896,83	1885,9	1881,67	1895	1884,5	1879	12,20	8,21	10,44	6,66%	6,05%	5,81%	5,60%	4,82%	4,76%
100on10Rp50Rs50_1	1136,3	1129,7	1128,37	1135	1130,5	1126	8,49	6,73	9,87	9,17%	8,53%	8,40%	7,70%	7,21%	6,93%
Média Total	**811,32**	**804,95**	**802,16**	**810,60**	**804,20**	**802,03**	**8,60**	**7,44**	**7,62**	**5,02%**	**3,19%**	**2,40%**	**-0,42%**	**-1,45%**	**-2,52%**

Tabela 5.3: Resultados obtidos pela aplicação do algoritmo GIVMP

GIVMP	**Média**			**Mediana**			**Desv Pad**			RPD_{avg}			RPD_{best}		
Instância	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**
I_50_10_S_1-124_1	112,067	110,9	109,833	112	110	109	4,03	3,20	3,28	-1,70%	-2,72%	-3,66%	-8,77%	-7,02%	-8,77%
I_50_15_S_1-124_1	74,6	73,2667	72,5	75	74	72	2,95	2,05	2,42	-0,53%	-2,31%	-3,33%	-9,33%	-6,67%	-9,33%
I_50_20_S_1-124_1	55,2667	52,6	53,7333	56	53	54	3,47	2,94	3,49	6,28%	1,15%	3,33%	-5,77%	-11,54%	-11,54%
I_100_10_S_1-124_1	229,767	222,9	222,6	228	223,5	221	8,39	6,08	7,40	3,97%	0,86%	0,72%	-3,17%	-3,62%	-4,52%
I_100_15_S_1-124_1	138,633	134,4	134,533	138	134,5	135	3,32	4,46	4,44	-1,68%	-4,68%	-4,59%	-6,38%	-9,22%	-11,35%
I_100_20_S_1-124_1	96,4667	92,8333	94,2	97,5	93	93,5	3,43	3,65	4,35	-3,53%	-7,17%	-5,80%	-9,00%	-13,00%	-14,00%
I_150_10_S_1-124_1	331,533	319,133	321,267	331	317	320,5	11,10	9,65	10,52	-0,14%	-3,88%	-3,23%	-6,63%	-9,94%	-9,34%
I_150_15_S_1-124_1	211,133	203,5	203,2	209	204	203,5	6,66	6,31	6,49	0,54%	-3,10%	-3,24%	-5,24%	-9,52%	-9,52%
I_150_20_S_1-124_1	135,733	126,667	127,533	135,5	125,5	127,5	4,95	5,63	3,94	-3,74%	-10,17%	-9,55%	-10,64%	-16,31%	-14,89%
80on4Rp50Rs50_1	2325,8	2300,53	2298,27	2324	2300,5	2300	13,83	9,77	8,75	5,96%	4,81%	4,70%	4,83%	3,92%	4,10%
80on6Rp50Rs50_1	1542,07	1533,5	1531	1542,5	1534	1531,5	9,51	8,76	8,45	7,84%	7,24%	7,06%	6,71%	6,22%	5,80%
80on10Rp50Rs50_1	916,133	908,9	910,667	913	907	909,5	10,31	6,60	9,93	9,10%	8,24%	8,45%	7,30%	7,18%	6,82%
100on4Rp50Rs50_1	2897,5	2864,17	2863,8	2897	2866	2864,5	17,42	14,99	13,25	5,78%	4,56%	4,55%	4,66%	3,39%	3,53%
100on6Rp50Rs50_1	1892,8	1874,07	1873	1891,5	1874	1871	10,47	6,63	7,82	6,44%	5,38%	5,32%	5,27%	4,82%	4,70%
100on10Rp50Rs50_1	1129,63	1122,77	1120,93	1128,5	1122	1120	7,92	7,02	6,52	8,52%	7,87%	7,69%	7,50%	6,73%	6,54%
Média Total	**805,94**	**796,01**	**795,80**	**805,23**	**795,87**	**795,50**	**7,85**	**6,52**	**6,74**	**2,87%**	**0,41%**	**0,56%**	**-1,91%**	**-3,64%**	**-4,12%**

GIVP. Nota-se, também, que o GIVMP apresentou desvios padrões pouco maiores que o IVP.

A visualização dos resultados é melhor realizada a partir do *boxplot* dos valores de RPD_{avg} para cada algoritmo (Figura 5.1). Nesse *boxplot*, percebe-se a semelhança entre os valores encontrados para o GIVMP e o IVP. Além do mais pode-se visualizar a superioridade do GIVMP ao analisar seu limite inferior, que é duas vezes menor que os limites do IVP e GIVP. Também é importante destacar a inferioridade do GIVP, pois 25% de seus valores de RPD_{avg} são equivalentes a 50% dos valores RPD_{avg} do GIVMP e do IVP.

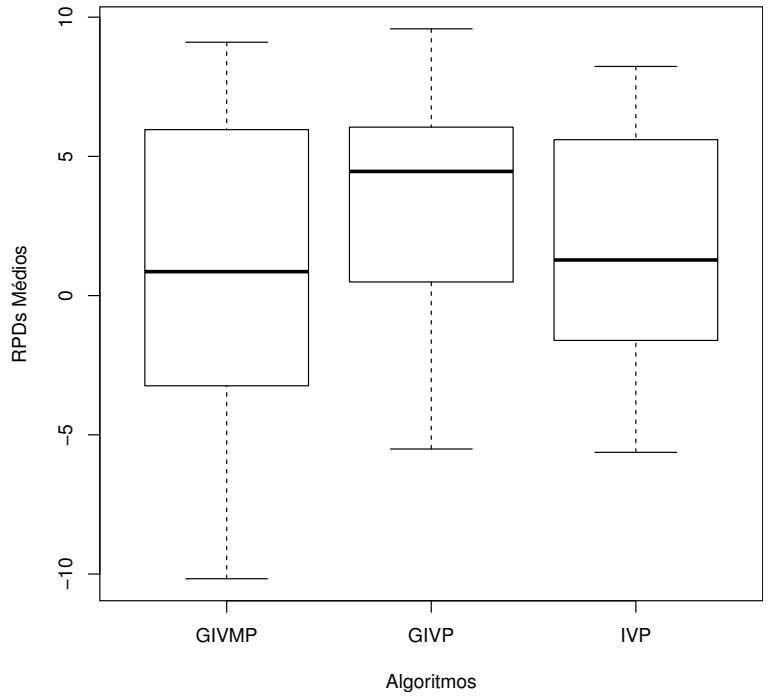

Figura 5.1: *Boxplot* mostrando os RPDs médios dos algoritmos IVP, GIVP e GIVMP

Levando em conta as análises feitas, tem-se indícios de que o algoritmo GIVMP é o melhor, seguido pelo IVP e o GIVP. Entretanto, na tentativa de validar esses indícios, foi realizada uma análise de variância - ANOVA (Montgomery, 2007) dos valores de RPD_{avg} encontrados pelos três algoritmos desenvolvidos. Pelo resultado do ANOVA não se pode dizer que existe diferença estatística entre os 3 algoritmos ao analisar os valores de RPD_{avg}, pois o p resultante da análise foi igual a $0,06$ que é maior que o $threshold = 0,05$.

Visando uma análise baseada no tempo que cada algoritmo leva para alcançar uma solução de qualidade foi realizado um teste de probabilidade empírica com os 3 algoritmos. A utilização deste teste é importante para se comparar diferentes algoritmos ou estratégias para resolver um problema, além do mais esse teste tem sido bastante utilizado como uma ferramenta de desenvolvimento de algoritmos e comparações (Feo et al., 1994; Ribeiro e Resende, 2011). Como o GIVMP possui uma limitação por tempo em seu processo iterativo para, ao final, executar o modelo PIM com o tempo restante, nesse teste o modelo PIM não foi acionado, utilizando somente o processo iterativo.

O teste de probabilidade empírica foi realizado com a instância *I_100_10_S_1-124_1* que contém 100 tarefas e 10 máquinas. O valor alvo escolhido para que cada algoritmo alcançasse foi 5% maior que a melhor solução conhecida. Todos os algoritmos foram aplicados 100 vezes e sempre que o alvo foi alcançado, eles eram interrompidos, registrando o tempo de execução em segundos. Ao final, os tempos marcados foram, então, ordenados de forma crescente tal que, para cada execução $i = 1, 2, ..., 100$ existe um tempo t_i e uma probabilidade $p_i = (i - 0,5)/100$ associada. O gráfico $t_i \times p_i$ gerado é apresentado na Figura 5.2. A fim de possibilitar uma comparação mais eficiente, as curvas de probabilidades empíricas foram sobrepostas.

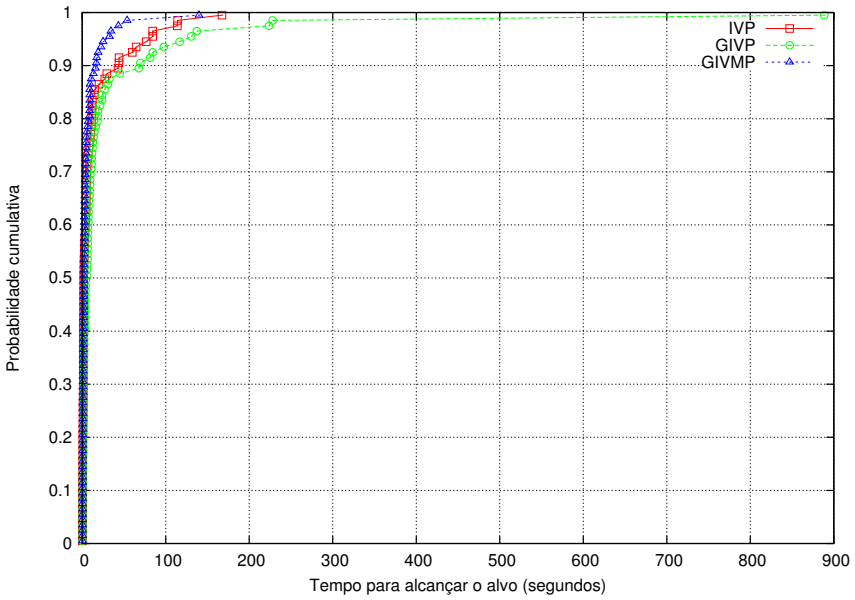

Figura 5.2: Distribuição empírica - I_100_10_S_1-124_1

Ao analisar as curvas de probabilidades empíricas pode-se perceber que o GIVMP mais uma vez superou os outros algoritmos, encontrando soluções de qualidade mais rapidamente que o GIVP e o IVP. O IVP teve um comportamento um pouco pior que o GIVMP. Já o GIVP novamente não obteve bons resultados levando mais tempo para alcançar o alvo.

Na Figura 5.2, observa-se ainda que o GIVMP foi o primeiro a alcançar o alvo desejado com uma probabilidade de 100% em pouco mais de 2 minutos. O IVP leva em torno de 3 minutos para alcançar o alvo com esta mesma probabilidade, 100%. Por outro lado, para o GIVP atingir 100% de chances de encontrar o valor alvo, são necessários quase 15 minutos.

Com o objetivo de validar a análise dos experimentos empíricos, foi realizado um experimento probabilístico de acordo com Ribeiro e Rosseti (2009) e Ribeiro et al. (2011). Sejam A1 e A2 dois algoritmos de busca estocásticos aplicados a um mesmo problema e sejam X1 e X2 as variáveis contínuas aleatórias que representam o tempo necessário para que os algoritmos A1 e A2, respectivamente, encontrem uma solução tão boa quanto o alvo desejado. Em Ribeiro e Rosseti (2009) foi desenvolvido uma ferramenta numérica para calcular a probabilidade do tempo de execução do algoritmo A1 ser menor ou igual ao tempo de execução do algoritmo A2, ou seja,

$P(X_1 \leq X_2)$.

Sejam IVP, GIVP e GIVMP iguais a A1, A2 e A3, respectivamente, e sejam X1, X2 e X3 as variáveis contínuas aleatórias que representam os tempos de execução desses algoritmos até encontrar o alvo. Ao utilizar tal ferramenta nos resultados obtidos pelos 3 algoritmos se tem a seguinte resposta, com relação à instância utilizada: a probabilidade de o IVP encontrar soluções próximas ao alvo em um tempo menor que o tempo gasto pelo GIVP é $P(X_1 \leq X_2) = 56,56\%$, com um erro de 4%. Já a probabilidade de o GIVMP ser mais rápido do que o GIVP é $P(X_3 \leq X_2) = 61,81\%$, também com um erro de 4%. Finalmente, ao se comparar o GIVMP com o IVP se tem $P(X_3 \leq X_1) = 50,89\%$, com um erro de 7%. Esse resultado confirma a análise feita sobre a Figura 5.2.

Levando todos os testes feitos em consideração, apesar de não se ter a comprovação de que o GIVMP é estatisticamente diferente e, consequentemente melhor que os outros dois (IVP e GIVP), existem fortes indícios de sua superioridade. Por isso, o GIVMP foi o algoritmo escolhido para ser aplicado nas instâncias de Vallada e Ruiz (2011) e Rabadi et al. (2006).

5.3 Instâncias de Vallada e Ruiz (2011)

Tomando-se o algoritmo GIVMP, foram escolhidas instâncias grandes disponíveis em SOA (2011) com as mesmas grandezas utilizadas nos testes da Seção 5.2 (com $n \in \{50, 100, 150\}$ e $m \in \{10, 15, 20\}$). Contudo, desta vez todas as instâncias de cada combinação de tarefas e máquinas foram utilizadas para os testes. Como são 40 instâncias para cada combinação e 9 combinações possíveis, no total foram utilizadas 360 instâncias.

Os parâmetros α de *geraSolucaoParcialmenteGulosa* e o parâmetro *vezesnivel* também foram os mesmos utilizados na Seção 5.2, sendo os valores 0,2 e 15, respectivamente. O critério de parada, dado pela Eq. (5.1) e utilizado em Vallada e Ruiz (2011), foi adotado com os seguintes valores de t: 10, 30 e 50. A métrica para a comparação é dada pela Eq. (5.2). Para cada instância foram realizadas 30 execuções do GIVMP, enquanto que em Vallada e Ruiz (2011) apenas 5 execuções foram realizadas. É importante ressaltar ainda que, a configuração do computador utilizado nesse trabalho foi semelhante à configuração utilizada em Vallada e Ruiz (2011), *PC Intel Core 2 Duo com 2.4 GHz e 2 GB de RAM* , o que possibilita comparações.

Na Tabela 5.4 estão presentes as médias dos desvios percentuais relativos, ou RPD_{avg}, encontrados para os algoritmos GA1, GA2 e GIVMP. Esses valores estão separados em colunas para cada valor de $t = 10, 30, 50$. Novamente aqui, os valores negativos de RPD representam soluções melhores do que os melhores valores encontrados em Vallada e Ruiz (2011) em seus testes.

Na Tabela 5.4, os valores em negrito indicam os melhores valores de RPD_{avg}. Nota-se que o GIVMP foi capaz de alcançar os melhores resultados. Na média, o GIVMP não só ganha em todos os conjuntos de instâncias, como também supera a maioria dos melhores resultados conhecidos até então. Para $t = 50$, os melhores valores conhecidos só não foram superados no conjunto de instâncias com 100 tarefas e 10 máquinas.

Tabela 5.4: Médias dos Desvios Percentuais Relativos (RPD_{avg}) dos algoritmos GA1, GA2 e GIVMP com $t = 10, 30, 50$

	GA1[1]			GA2[1]			GIVMP		
Instâncias	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**
50x10	13,56%	12,31%	11,66%	7,79%	6,92%	6,49%	**1,79%**	**0,58%**	**-0,11%**
50x15	13,87%	13,95%	12,74%	12,25%	8,92%	9,20%	**-0,39%**	**-2,10%**	**-2,57%**
50x20	12,92%	12,58%	13,44%	11,08%	8,04%	9,57%	**3,92%**	**1,43%**	**-0,05%**
100x10	13,11%	10,46%	9,68%	15,72%	6,76%	5,54%	**2,66%**	**0,99%**	**0,42%**
100x15	15,41%	13,95%	12,94%	22,15%	8,36%	7,32%	**-0,11%**	**-1,88%**	**-2,74%**
100x20	15,34%	13,65%	13,60%	22,02%	9,79%	8,59%	**-1,66%**	**-3,65%**	**-4,60%**
150x10	10,95%	8,19%	7,69%	18,40%	5,75%	5,28%	**2,11%**	**0,14%**	**-0,70%**
150x15	14,51%	11,93%	11,78%	24,89%	8,09%	6,80%	**1,08%**	**-1,03%**	**-1,78%**
150x20	13,82%	12,66%	12,49%	22,63%	9,53%	7,40%	**-1,91%**	**-3,90%**	**-4,75%**
Média Total	13,72%	12,19%	11,78%	17,44%	8,02%	7,35%	**0,83%**	**-1,05%**	**-1,87%**

[1]Testes realizados em um PC Intel Core 2 Duo com 2.4 GHz e 2 GB de RAM

A Figura 5.3 mostra o *boxplot* de todos os valores de RPD_{avg} encontrados em cada algoritmo (GA1, GA2 e GIVMP) para cada instância e com t igual a 10, 30 e 50. Pela análise da figura, nota-se a hegemonia dos valores encontrados pelo GIVMP, os quais em sua totalidade (100%) são melhores que qualquer valor de GA1 ou GA2. Outro ponto importante é que mais de 50% dos valores RPD_{avg} obtidos pelo GIVMP são melhores que as melhores soluções de Vallada e Ruiz (2011).

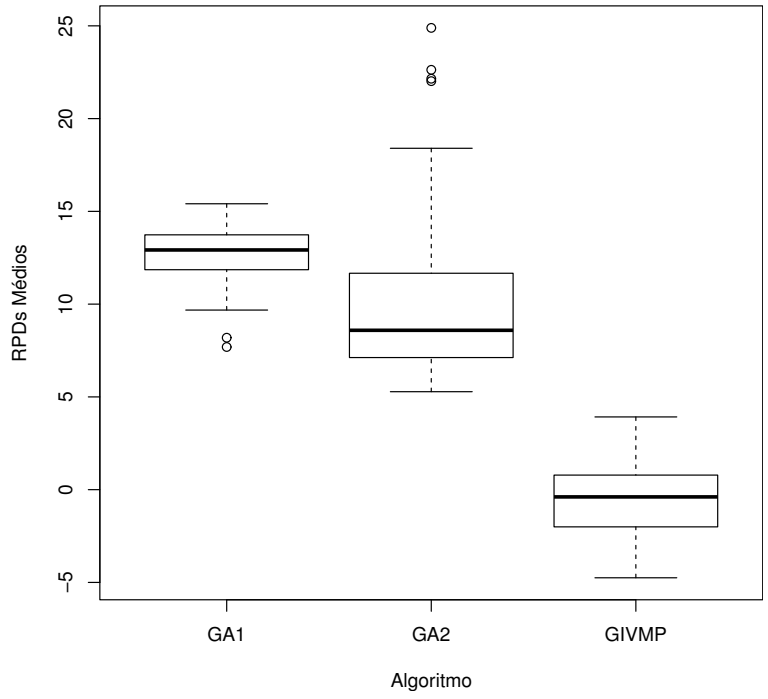

Figura 5.3: *Boxplot* mostrando os RPDs médios dos algoritmos GA1, GA2 e GIVMP

Mais uma vez, busca-se a comprovação estatística de que existe diferença entre os algoritmos em questão (GA1, GA2 e GIVMP). Sendo assim, foi realizado um ANOVA que resultou em um valor $p = 2,2 \times 10^{-16}$. Como p é menor que o $threshold = 0,05$, significa que existe diferença estatística entre os algoritmos.

Com a intenção de verificar onde estão essas diferenças, foi usado um teste de

Tukey HSD, com nível de confiança de 95% e $threshold = 0,05$. A Tabela 5.5 contém as diferenças nos valores médios de RPD (diff), o limite inferior (lwr), o limite superior (upr) e o valor p (p) para cada par de algoritmos. Pode ser visto pelos valores de p que o GIVMP se difere estatisticamente tanto do GA1 quanto do GA2, por conta desses valores serem menores que o *threshold*. Por outro lado, os algoritmos genéticos, GA1 e GA2, não são diferentes estatisticamente.

Tabela 5.5: Resultados do teste Tukey HSD

Algoritmos	diff	lwr	upr	p
GA2-GA1	-1,626296	-4,083793	0,8312008	0,2598164
GIVMP-GA1	-13,259259	-15,716756	-10,8017622	0,0000000
GIVMP-GA2	-11,632963	-14,090460	-9,1754659	0,0000000

Plotando os resultados obtidos pelo teste Tukey HSD (Figura 5.4, pode ser observado que o GIVMP é diferente estatisticamente do GA1 e GA2, pois comparando com cada um deles, o gráfico não passa pelo ponto zero. Tal fato não se aplica quando há a comparação do GA1 com o GA2, pois o gráfico passa pelo ponto zero, o que mostra que eles não possuem diferença estatística.

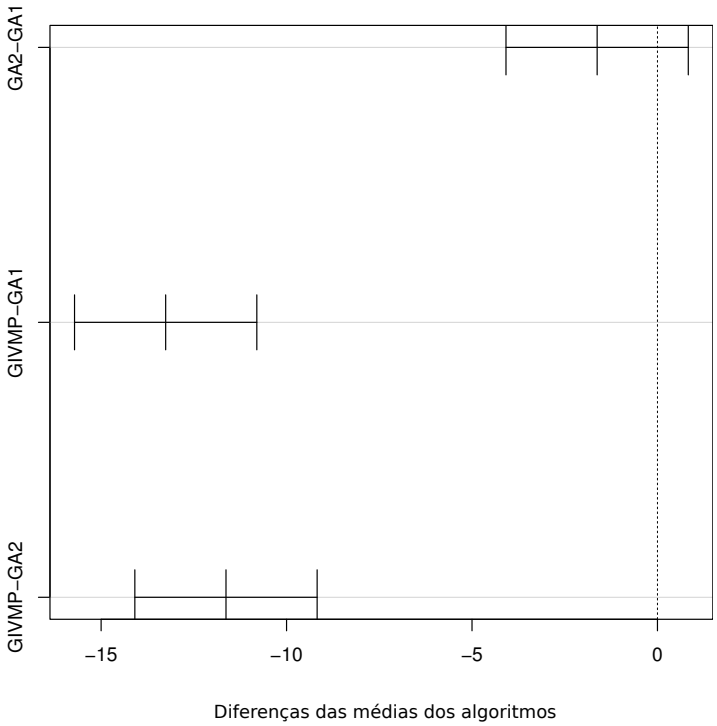

Figura 5.4: Resultados gráficos do teste Tukey HSD

Desta forma, pode-se concluir que para estas instâncias e, com base em uma análise estatística, que o GIVMP é o melhor algoritmo. Ser o melhor algoritmo significa que ele é capaz de obter as melhores soluções para o problema em questão, o UPMSPST.

Em DECOM (2012) são disponibilizados todos os resultados alcançados pelo GIVMP, incluindo, para cada instância, a pior solução, a melhor solução, o valor médio, a mediana, o desvio percentual relativo médio e o desvio percentual em relação à melhor solução conhecida. Os resultados completos para cada grupo de instâncias grandes e pequenas também são apresentados no Apêndice B.

5.4 Instâncias de Rabadi et al. (2006)

O GIVMP também foi aplicado às instâncias grandes, desenvolvidas em Rabadi et al. (2006), utilizadas pelos testes da Seção 5.2 (com $n = \{80, 100\}$ e $m = \{4, 6, 10\}$). Porém, ao invés de utilizar apenas uma instância de cada combinação de máquinas e tarefas, foram utilizadas todas as 45 instâncias de cada combinação desta, totalizando 270 instâncias.

Os parâmetros do GIVMP utilizados foram idênticos a todos os testes, $\alpha = 0,2$ e $vezesnivel = 15$. O critério de parada, todavia, não foi o mesmo, isso se deve ao fato de que tal critério não é estabelecido por nenhum trabalho relacionado a essas instâncias. Todos os trabalhos que estudam tais instâncias utilizam critérios de parada específicos de cada método proposto. Como o GIVMP é limitado por tempo, tomou-se os melhores tempos encontrados na literatura, os tempos médios obtidos pelo MVND (Fleszar et al., 2011). Outra mudança importante ao trabalhar com essas instâncias é ao comparar os resultados, pois a grande maioria dos trabalhos que tratam destes problemas-teste apresentam como resultado o RPD_{best}, que é o desvio percentual relativo da melhor solução encontrada pelo algoritmo. Esse desvio é calculado em relação ao limite inferior (LB) da Eqs. (5.3), (5.4), e (5.5). O GIVMP foi executado 30 vezes para cada instância e o RPD_{best} foi calculado a partir da melhor solução encontrada nessas 30 execuções.

A Tabela 5.6 mostra as médias dos RPD_{best} encontrados para os algoritmos Meta-RaPS (Rabadi et al., 2006), ACO (Arnaout et al., 2010), RSA (Ying et al., 2010), MVND (Fleszar et al., 2011) e GIVMP. Também são encontrados nessa tabela os tempo médios, em segundos, que cada algoritmo foi executado, com exceção do Meta-RaPS por conta de seus tempos não terem sido divulgados. Observa-se que os tempos do GIVMP são iguais aos tempos do MVND, pois foram usados para propiciar a comparação.

A Tabela 5.6 apresenta os melhores resultados de RPD_{best} marcados em negrito. Como pode-se perceber, o MVND encontrou melhores soluções para todos os conjuntos de instâncias. Contudo, nota-se que o GIVMP supera os outros três algoritmos (Meta-RaPS, ACO e RSA) em todos os conjuntos de instâncias. Apesar de a comparação de tempo não ser justa, em virtude das diferentes configurações das máquinas usadas, observa-se que o GIVMP produziu soluções de qualidade em um tempo muitíssimo inferior.

A Figura 5.5 mostra os valores RPD_{best} em um *boxplot* para cada algoritmo. A evidência da superioridade do MVND pode ser vista nessa figura, pois ele tem 50% de seus valores melhores que todos os algoritmos, exceto o GIVMP que, por sua vez supera os outros algoritmos com 25% de seus valores de RPD_{best}.

Novamente, recorreu-se à estatística para saber se existe diferença entre os algoritmos analisados (Meta-RaPS, ACO, RSA, MVND e GIVMP). Foi realizado um

Tabela 5.6: Médias dos RPD_{best} dos algoritmos Meta-RaPS, ACO, RSA, MVND e GIVMP

	Meta-RaPS[1]	**ACO**[2]		**RSA**[3]		**MVND**[4]		**GIVMP**	
Instâncias	RPD_{best}	RPD_{best}	**Tempo (s)**	RPD_{best}	**Tempo (s)**	RPD_{best}	**Tempo (s)**	RPD_{best}	**Tempo (s)**
80 x 4	4,41%	3,26%	311,4	3,16%	68,5	**1,98%**	8,2	2,70%	8,2
80 x 6	6,19%	7,29%	-	5,95%	67,2	**4,66%**	13	5,37%	13
80 x 10	5,89%	8,01%	-	5,76%	101,1	**3,97%**	6,5	5,57%	6,5
100 x 4	3,39%	2,91%	557,4	2,92%	120,9	**1,63%**	19,2	2,28%	19,2
100 x 6	4,79%	5,14%	544,2	4,38%	144,7	**2,81%**	21,4	3,86%	21,4
100 x 10	6,43%	6,85%	-	5,07%	220,3	**3,28%**	14	4,88%	14
Média Total	5,18%	5,58%	471,00	5,58%	120,45	**3,06%**	13,72	4,11%	13,72

[1]Testes realizados em um PC Pentium IV com 1.7 GHz

[2]Testes realizados em um PC Pentium IV com 2GB de RAM

[3]Testes realizados em um PC Pentium IV com 1.5 GHz e 512 MB de RAM

[4]Testes realizados em um Dell Optiplex 760, Intel Core 2 Quad Q8200 com 2.33 GHz

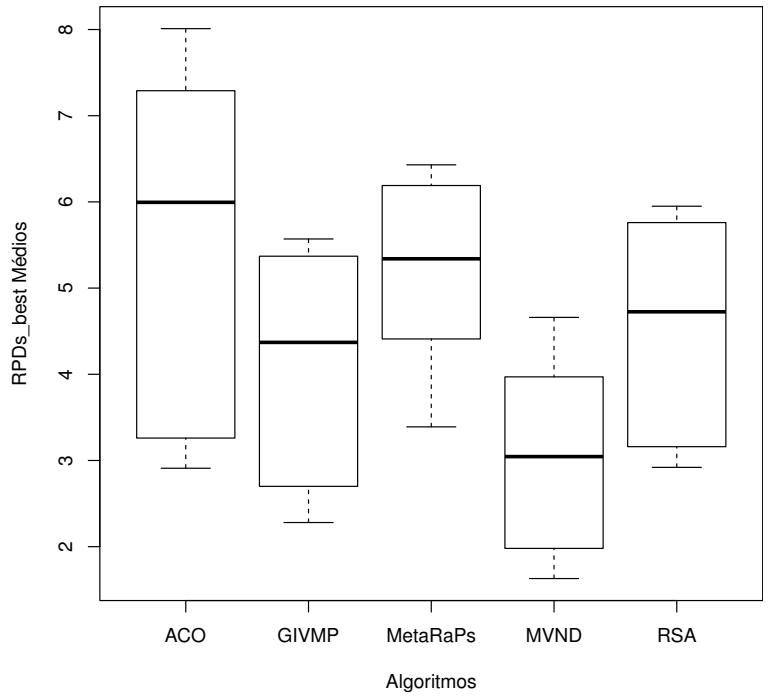

Figura 5.5: *Boxplot* mostrando os RPD_{best} dos algoritmos Meta-RaPS, ACO, RSA, MVND e GIVMP

ANOVA que resultou em um valor $p = 0,057$. Como p é maior que o $threshold = 0,05$, significa que não existe diferença estatística entre os algoritmos para esta medida de comparação e para estas instâncias.

Finalmente, é importante ressaltar que a comparação dos algoritmos apenas pelos melhores valores obtidos não é eficiente, quando se busca um algoritmo robusto para um problema. De forma a permitir comparações futuras com o algoritmo desenvolvido, são disponibilizados em DECOM (2012) todos os resultados alcançados pelo GIVMP, incluindo, para cada instância, a pior solução, a melhor solução, o valor médio, a mediana, o desvio percentual relativo médio e o desvio percentual em relação à melhor solução conhecida. Os resultados completos para cada grupo de instâncias grandes e pequenas também são apresentados no Apêndice B.

Capítulo 6

Conclusões e Trabalhos Futuros

A Seção 6.1 apresenta as conclusões feitas a partir do desenvolvimento deste trabalho. A Seção 6.2 indica trabalhos que ainda podem ser desenvolvidos.

6.1 Conclusões

Este trabalho estudou o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (UPMSPST). Tal problema busca encontrar um sequenciamento de tarefas em máquinas não-relacionadas visando a minimização do tempo máximo de conclusão do sequenciamento, conhecido como *makespan*. As máquinas são não-relacionadas, pois o tempo de processamento de uma mesma tarefa difere para cada máquina. No UPMSPST também existem tempos de preparação entre as tarefas que são dependentes da máquina e da sequência de tarefas em cada máquina.

Devido ao fato de que o UPMSPST pertence à classe NP-difícil, instâncias de maior porte podem requerer tempos de processamento inviáveis para sua resolução exata. Este fato motiva o desenvolvimento de algoritmos heurísticos, buscando encontrar soluções de qualidade para o UPMSPST. A união de várias heurísticas eficientes, o que caracteriza um algoritmo heurístico híbrido, contribui para a formação de um algoritmo robusto. Assim sendo, foram propostos três algoritmos heurísticos híbridos para resolver o UPMSPST.

O primeiro algoritmo criado, chamado IVP, é baseado no *Iterated Local Search* (ILS). As perturbações do ILS consistem em realocar tarefas de uma máquina para outra, inserindo-as em uma posição que contribua para que a máquina tenha um tempo de conclusão menor. A solução inicial é gerada por uma heurística de construção gulosa, de nome *Adaptive Shortest Processing Time* (ASPT). As buscas locais do ILS são realizadas pelo método *Variable Neighborhood Descent* (VND), explorando o espaço de busca através de buscas locais, aplicadas de forma ordenada, que utilizam movimentos de inserções e trocas de tarefas. Também foi incluído, nesse algoritmo, acionamentos da técnica *Path Relinking* (PR) com a estratégia *backward relinking* após as buscas locais, objetivando intensificar e diversificar a busca.

O GIVP, que foi o segundo algoritmo implementado, é considerado como um aperfeiçoamento do IVP ao substituir a construção gulosa pela fase de construção do método *Greedy Randomized Adaptive Search Procedure* (GRASP).

O terceiro e último algoritmo desenvolvido foi nomeado GIVMP. Este, por sua vez, é um aperfeiçoamento do GIVP, pois além de substituir a estratégia *backward relinking* pela estratégia *mixed relinking* e o VND com ordem para as buscas locais pelo VND sem ordem definida para as buscas locais, há também a incorporação de um modelo de programação inteira mista (PIM) que tenta otimizar a máquina que contém o *makespan*.

Os três algoritmos implementados foram aplicados a um conjunto pequeno de problemas-teste presentes em SchedulingResearch (2005) e em SOA (2011). Os resultados mostraram que o GIVMP foi capaz de superar os outros dois algoritmos, IVP e GIVP, pois não só conseguiu gerar soluções melhores, mas também consumiu menos tempo de processamento que os outros.

Após a definição do GIVMP como melhor algoritmo, dentre os três, o mesmo foi aplicado a um conjunto maior de instâncias disponíveis em SOA (2011) e SchedulingResearch (2005). Ao aplicar o GIVMP nas instâncias disponíveis em SOA (2011), os resultados foram comparados a dois algoritmos genéticos (Vallada e Ruiz, 2011). Foram feitas análises estatísticas que comprovaram que GIVMP é melhor que ambos, conseguindo ter uma baixa variabilidade e melhorando as melhores soluções conhecidas até então. Nas instâncias disponíveis em SchedulingResearch (2005), o GIVMP foi comparado aos algoritmos Meta-RaPS (Rabadi et al., 2006), ACO (Arnaout et al., 2010), RSA (Ying et al., 2010) e MVND (Fleszar et al., 2011). Apesar de não se comprovar a diferença estatística entre esses algoritmos com relação à métrica utilizada (média dos desvios percentuais relativos dos melhores valores obtidos), o GIVMP tem o segundo melhor desempenho. Observa-se, no entanto, que neste segundo conjunto de problemas-teste, não foram disponibilizados os desvios percentuais relativos médios dos algoritmos, mas apenas os desvios dos melhores valores encontrados, impedindo uma comparação de robustez dos algoritmos. Neste sentido, foram disponibilizados, para comparação futura, todos os resultados alcançados pelo algoritmo proposto.

Em vista dessas análises, pode-se concluir que o algoritmo GIVMP é superior às versões IVP e GIVP, assim como aos algoritmos GA1, GA2, Meta-RaPS, ACO e RSA. Trata-se, portanto, de um algoritmo eficiente que supera vários outros algoritmos do estado da arte na resolução de problemas de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. O GIVMP só não capaz de superar o algoritmo MVND, mas há ainda estratégias que não foram testadas a serem a ele incorporadas que podem ser suficientes para melhorar seu desempenho.

6.2 Trabalhos Futuros

Para melhorar o desempenho do algoritmo GIVMP, de forma a superar o algoritmo MVND, são apontados os seguintes trabalhos futuros:

- avaliar a contribuição de cada módulo utilizado pelo GIVMP, a saber:
  - RVND
  - Mixed Path Relinking

  - Modelo de Programação Inteira Mista

- avaliar a inclusão de outras estratégias de vizinhança, como por exemplo, a utilização de movimentos envolvendo a realocação de $k \geq 2$ tarefas de uma máquina para outra ou para outra posição em uma mesma máquina.

- explorar técnicas de programação inteira para resolução exata do maior número possível de instâncias ainda consideradas difíceis para esses métodos.

Referências Bibliográficas

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

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

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling.* John Wiley & Sons.

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

Cheng, TCE e Sin, CCS. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, v. 47, n. 3, p. 271–292.

de Paula, M. R.; Ravetti, M. G.; Mateus, G. R. e Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, v. 18, p. 101–115.

DECOM„ (2012). Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times. Página que contém resultados para o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. Disponível em `http://www.decom.ufop.br/prof/marcone/projects/upmsp.html`. Acesso em 20 de janeiro de 2012.

Feo, T. e Resende, M. (1995). Greedy randomized search procedure. *Journal of Global Optimization*, v. 6, p. 109–133.

Feo, T.A.; Resende, M.G.C. e Smith, S.H. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, v. 42, p. 860–878.

Fleszar, K.; Charalambous, C. e Hindi, K.S. (2011). A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, v. . doi:10.1007/s10845-011-0522-8.

Frinhani, R. M. D. (2011). Grasp com path-relinking para agrupamento de dados biológicos. Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, UFMG, Belo Horizonte, Brasil.

Garey, M.R e Johnson, D.S. (1979). Computers and intractability: A guide to the theory of np-completeness. *WH Freeman & Co., San Francisco*, v. 174.

Glover, F. (1996). Tabu search and adaptive memory programming - advances, applications and challenges. Barr, R. S.; Helgason, R. V. e Kennington, J. L., editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, p. 1–75. Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Berkeley.

Gomes, H. A. S. *Utilização da Metaheurística Simulated Annealing no Problema de Alocação de Pessoal em Empresas de Transporte Coletivo por Ônibus.* PhD thesis, Universidade Federal do Ceará, (2003).

Graham, R.L.; Lawler, E.L.; Lenstra, J.K. e Kan, A.H.G.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete Mathematics*, v. 5, n. 2, p. 287–326.

Guinet, A. (1991). Textile production systems: a succession of non-identical parallel processors shops. *Journal of Operational Research Society*, v. 42, n. 8, p. 655–671.

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

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

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press.

Karp, Richard M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, v. 40, n. 4, p. 85–103.

Kim, D. W.; Kim, K. H.; Jang, W. e Frank Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, v. 18, p. 223–231.

Kim, D. W.; Na, D. G. e Frank Chen, F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, v. 19, p. 173–181.

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

Laporte, G. (1992). The vehicle-routing problem - an overview of exact and approximate algorithms. *European Journal of Operational Research*, v. 59, p. 345–358.

Logendran, R.; McDonell, B. e Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations research*, v. 34, n. 11, p. 3420–3438.

Lourenço, H. R.; Martin, O. e Stützle, T. (2003). Iterated local search. Glover, F. e Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 321–353. Kluwer Academic Publishers, Norwell, MA.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, v. 6, n. 1, p. 1–12.

Mladenovic, Nenad e Hansen, Pierre. (1997). Variable neighborhood search. *Computers and Operations Research*, v. 24, n. 11, p. 1097–1100.

Montgomery, D. (2007). *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, fifth edição.

Nogueira, J. P. C. M. (2011). Heurísticas híbridas para o problema de programação de tarefas em máquinas paralelas não relacionadas com penalidades por antecipação e atraso. Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, UFV, Viçosa, Brasil.

Öncan, T.; Altinel, I.K. e Laporte, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research*, v. 36, n. 3, p. 637–654.

Pereira Lopes, M. J. e de Carvalho, J. M. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, v. 176, p. 1508–1527.

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

Randall, P. A. e Kurz, M. E. (2007). Effectiveness of Adaptive Crossover Procedures for a Genetic Algorithm to Schedule Unrelated Parallel Machines with Setups. *International Journal of Operational Research*, v. 4, p. 1–10.

Resende, M.G.C. e Ribeiro, C.C. (2005). Grasp with path-relinking: Recent advances and applications. *Metaheuristics: progress as real problem solvers*, v. 1, p. 29–63.

Resende, M.G.C. e Ribeiro, C.C. (2010). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. *Handbook of metaheuristics*, v. 146, p. 283–319.

Ribeiro, C. C. (1996). Metaheuristics and applications. In *Advanced School on Artificial Intelligence*, Estoril, Portugal, v. .

Ribeiro, C. C.; Uchoa, E. e Werneck, R. F. (2002). A hybrid grasp with perturbations for the steiner problem in graphs. *INFORMSJC*, v. 14, p. 228–246.

Ribeiro, C.C. e Resende, M.G.C. (2011). Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics*, v. , p. 1–22.

Ribeiro, C.C. e Rosseti, I. (2009). Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Proceedings of the VIII Metaheuristics International Conference*, Hamburg.

Ribeiro, C.C.; Rosseti, I. e Vallejos, R. (2011). Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, v. . Accepted for publication. Available at http://www.ic.uff.br/∼celso/artigos/runtimes.pdf.

Rossetti, I. C. M. (2003). Heurísticas para o problema de síntese de redes a 2 caminhos. Tese de doutorado, Programa de Pós-Graduação em Informática, PUC-RJ, Rio de Janeiro, Brasil.

Sarin, S.C.; Sherali, H.D. e Bhootra, A. (2005). New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations research letters*, v. 33, n. 1, p. 62–70.

SchedulingResearch„ (2005). Scheduling Research Virtual Center. A web site that includes benchmark problem data sets and solutions for scheduling problems. Disponível em `http://www.schedulingresearch.com`. Acesso em 3 de dezembro de 2011.

SOA„ (2011). Sistemas de Optimización Aplicada. Página que contém problemas-teste para o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. Disponível em `http://soa.iti.es/problem-instances`. Acesso em 30 de novembro de 2011.

Souza, M.J.F.; Coelho, I.M.; Ribas, S.; Santos, H.G. e Merschmann, L.H.C. (2010). A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, v. 207, n. 2, p. 1041–1051.

Subramanian, A.; Drummond, LMA; Bentes, C.; Ochi, LS e Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, v. 37, n. 11, p. 1899–1911.

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

Weng, M. X.; Lu, J. e Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, v. 70, p. 215–226.

Ying, Kuo-Ching; Lee, Zne-Jung e Lin, Shih-Wei. (2010). Makespan minimisation for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, v. . doi:10.1007/s10845-010-0483-3.

Apêndice A

Publicações

A seguir são listados os trabalhos oriundos desta dissertação que foram apresentados e aceitos para publicação em anais de eventos:

1. **Título**: Algoritmos Genéticos para o Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência
   Autores: Haddad, Matheus Nohra; Souza, Marcone Jamilson Freitas; Santos, Haroldo Gambini
   Evento: XLIII Simpósio Brasileiro de Pesquisa Operacional (XLIII SBPO)
   Local: Ubatuba (SP)
   Período: 15 a 18 de agosto de 2011

2. **Título**: Um Algoritmo Baseado em *Iterated Local Search* para o Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência
   Autores: Haddad, Matheus Nohra; Souza, Marcone Jamilson Freitas; Santos, Haroldo Gambini; Silva, Leandro Augusto de Araújo
   Evento: X Congresso Brasileiro de Inteligência Computacional (X CBIC)
   Local: Fortaleza (CE)
   Período: 8 a 11 de novembro de 2011

3. **Título**: A hybrid algorithm for solving the unrelated parallel machine scheduling problem with sequence dependent setup times
   Autores: Haddad, Matheus Nohra; Souza, Marcone Jamilson Freitas; Castro, Maria Cláudia Feres Monteiro; Coelho, Vitor Nazário
   Evento: XXXII Iberian Latin American Congress on Computational Methods in Engineering (XXXII CILAMCE)
   Local: Ouro Preto (MG)
   Período: 13 a 16 de novembro de 2011

Apêndice B

Resultados Completos

São apresentados, a seguir, os resultados completos da aplicação do algoritmo GIVMP aos conjuntos de instâncias disponíveis em SchedulingResearch (2005) e SOA (2011).

Na Tabela B.1 são listados, para cada grupo de instâncias, a média do valor da pior solução obtida, a média do valor da melhor solução, a média do valor médio das execuções, a média da mediana, assim como a média dos desvios percentuais relativos das melhores soluções em relação aos limites inferiores (RPD^{LB}_{best}), a média dos desvios percentuais relativos dos valores médios em relação aos limites inferiores (RPD^{LB}_{avg}) e média do tempo de execução. Na Tabela B.2 também são listados estes mesmos valores, porém o tempo de execução foi calculado pela Eq. (5.1), com $t = 30$. Outra diferença, desta tabela, é a inclusão das média dos desvios percentuais relativos das melhores soluções em relação aos valores ótimos (RPD^{Opt}_{best}) e das médias dos desvios percentuais relativos dos valores médios em relação aos valores ótimos (RPD^{Opt}_{avg}).

Nas tabelas B.3, B.4, B.5 e B.6 são listados, para cada grupo de instâncias e para cada valor de t (10, 30 e 50), a média do valor da pior solução obtida, a média do valor da melhor solução, a média do valor médio das execuções, a média da mediana, assim como a média dos desvios percentuais relativos das melhores soluções em relação aos melhores valores disponíveis em SOA (2011), RPD_{best}, e a média dos desvios percentuais relativos dos valores médios em relação aos melhores valores disponíveis em SOA (2011), RPD_{avg}.

O cálculo do tempo de execução envolvendo o parâmetro t é obtido pela Eq. (5.1), presente na página 51. O cálculo dos desvios percentuais relativos é feito conforme definido na Eq. (5.2) da página 51. O cálculo do limites inferiores são definidos pelas Eqs. (5.3), (5.4), e (5.5), conforme visto na página 51. Todas as tabelas foram obtidas a partir de 30 execuções do GIVMP a cada instância.

Tabela B.1: Resultados do GIVMP nas instâncias grandes disponíveis em SchedulingResearch (2005)

Instâncias	**Pior Solução**	**Melhor Solução**	**Média**	**Mediana**	$RPD_{best}^{LB\,1}$	$RPD_{avg}^{LB\,2}$	**Tempo (s)**
80 x 4	3338,44	3281,38	3305,09	3303,83	2,70%	3,48%	8,2
80 x 6	2244,89	2205,24	2223,37	2222,99	5,37%	6,29%	13
80 x 10	1332,42	1298,91	1312,10	1311,30	5,57%	6,70%	6,5
100 x 4	4141,78	4081,78	4106,31	4104,36	2,28%	2,94%	19,2
100 x 6	2753,18	2712,20	2730,50	2729,62	3,86%	4,61%	21,4
100 x 10	1648,93	1612,18	1626,79	1625,87	4,88%	5,89%	14
Média Total	**2576,61**	**2531,95**	**2550,69**	**2549,66**	**4,11%**	**4,99%**	**13,72**

[1]Média dos desvios percentuais relativos das melhores soluções em relação aos limites inferiores (LB)

[2]Média dos desvios percentuais relativos dos valores médios em relação aos limites inferiores (LB)

Tabela B.2: Resultados do GIVMP em todas as instâncias pequenas disponíveis em SchedulingResearch (2005), com t = 30

Instância	**Valor Ótimo**[1]	**Melhor**	**Pior**	**Média**	**Mediana**	RPD_{best}^{Opt2}	RPD_{avg}^{Opt3}	RPD_{best}^{LB4}	RPD_{avg}^{LB5}
6 x 2	551,58	551,58	551,58	551,58	551,58	0,00%	0,00%	7,69%	7,69%
7 x 2	688,42	688,42	688,42	688,42	688,42	0,00%	0,00%	14,43%	14,43%
8 x 2	721,49	721,49	721,53	721,50	721,49	0,00%	0,00%	5,82%	5,82%
9 x 2	858,56	858,56	858,71	858,57	858,56	0,00%	0,00%	11,48%	11,48%
10 x 2	-	888,40	888,71	888,45	888,40	-	-	4,76%	4,77%
11 x 2	-	1013,07	1013,58	1013,25	1013,22	-	-	8,70%	8,72%
6 x 4	346,60	346,60	354,82	349,17	347,39	0,00%	0,74%	40,20%	41,29%
7 x 4	353,78	353,78	354,69	354,00	353,96	0,00%	0,06%	23,03%	23,13%
8 x 4	364,51	364,51	364,60	364,54	364,51	0,00%	0,01%	12,06%	12,06%
9 x 4	-	493,67	493,67	493,67	493,67	-	-	34,42%	34,42%
10 x 4	-	508,89	509,13	508,91	508,89	-	-	24,22%	24,23%
11 x 4	-	516,29	518,58	517,06	516,82	-	-	15,64%	15,81%
8 x 6	-	334,87	349,78	339,16	338,13	-	-	61,97%	64,13%
9 x 6	-	340,31	343,16	340,66	340,38	-	-	45,50%	45,69%
10 x 6	-	344,78	347,58	345,23	344,98	-	-	33,50%	33,68%
11 x 6	-	349,96	351,13	350,21	350,09	-	-	23,41%	23,51%
10 x 8	-	326,13	347,02	332,87	331,77	-	-	64,70%	68,30%
11 x 8	-	330,64	339,82	333,33	332,46	-	-	52,66%	53,87%

[1]Média dos valores ótimos encontrados pelo CPLEX

[2]Média dos desvios percentuais relativos das melhores soluções em relação aos valores ótimos

[3]Média dos desvios percentuais relativos dos valores médios em relação aos valores ótimos

[4]Média dos desvios percentuais relativos das melhores soluções em relação aos limites inferiores (LB)

[5]Média dos desvios percentuais relativos dos valores médios em relação aos limites inferiores (LB)

Tabela B.3: Resultados do GIVMP nas instâncias grandes disponíveis em SOA (2011), com t = 10, 30 e 50

t = 10						
Instâncias	**Melhor**	**Pior**	**Média**	**Mediana**	RPD_{best}	RPD_{avg}
50x10	93,85	105,15	99,07	99,04	-3,35%	1,79%
50x15	54,40	64,15	58,26	57,95	-6,89%	-0,39%
50x20	36,68	49,05	41,05	40,60	-7,03%	3,92%
100x10	171,08	192,48	181,16	181,05	-2,78%	2,66%
100x15	99,75	114,75	106,54	106,28	-6,29%	-0,11%
100x20	67,25	80,08	72,82	72,59	-9,15%	-1,66%
150x10	247,40	275,58	259,38	259,29	-2,33%	2,11%
150x15	143,78	164,48	152,81	152,46	-4,64%	1,08%
150x20	96,88	112,88	104,19	103,94	-8,65%	-1,91%
Média Total	**112,34**	**128,73**	**119,48**	**119,24**	**-5,68%**	**0,83%**
t = 30						
Instâncias	**Melhor**	**Pior**	**Média**	**Mediana**	RPD_{best}	RPD_{avg}
50x10	93,03	103,75	97,80	97,51	-4,16%	0,58%
50x15	53,93	62,65	57,29	57,05	-7,68%	-2,10%
50x20	36,03	46,85	40,02	39,59	-8,64%	1,43%
100x10	168,43	188,28	177,94	178,00	-4,12%	0,99%
100x15	98,05	112,45	104,52	104,23	-7,77%	-1,88%
100x20	66,23	77,73	71,31	71,06	-10,36%	-3,65%
150x10	241,03	269,60	253,97	253,51	-4,66%	0,14%
150x15	140,65	159,38	149,31	149,00	-6,61%	-1,03%
150x20	95,38	109,93	101,93	101,78	-9,90%	-3,90%
Média Total	**110,30**	**125,62**	**117,12**	**116,86**	**-7,10%**	**-1,05%**
t = 50						
Instâncias	**Melhor**	**Pior**	**Média**	**Mediana**	RPD_{best}	RPD_{avg}
50x10	92,70	102,70	97,11	96,90	-4,44%	-0,11%
50x15	53,83	62,48	56,98	56,63	-7,80%	-2,57%
50x20	35,83	46,03	39,41	38,95	-9,18%	-0,05%
100x10	167,73	186,93	176,89	176,59	-4,50%	0,42%
100x15	97,73	110,45	103,58	103,41	-8,06%	-2,74%
100x20	65,60	77,28	70,59	70,35	-11,28%	-4,60%
150x10	239,50	266,45	251,67	251,40	-5,22%	-0,70%
150x15	139,28	158,78	148,19	148,04	-7,38%	-1,78%
150x20	94,55	109,50	101,01	100,80	-10,61%	-4,75%
Média Total	**109,64**	**124,51**	**116,16**	**115,90**	**-7,61%**	**-1,87%**

Tabela B.4: Resultados do GIVMP em todas as instâncias pequenas disponíveis em SOA (2011), com t = 10

Instância	Melhor	Pior	Média	Mediana	RPD_{best}	RPD_{avg}
10 x 2	231,03	231,30	231,12	231,03	0,00%	0,03%
10 x 3	130,08	130,15	130,10	130,08	0,00%	0,01%
10 x 4	97,88	98,20	98,00	97,96	0,00%	0,13%
10 x 5	71,38	72,45	71,69	71,55	0,00%	0,45%
11 x 2	265,13	266,15	265,39	265,23	0,00%	0,10%
11 x 3	156,63	156,63	156,63	156,63	0,00%	0,00%
11 x 4	108,23	108,70	108,41	108,41	0,00%	0,16%
11 x 5	78,53	79,23	78,84	78,83	-0,16%	0,28%
6 x 2	143,35	143,35	143,35	143,35	0,00%	0,00%
6 x 3	88,50	89,13	88,87	88,68	0,11%	0,47%
6 x 4	59,75	65,33	62,08	61,91	0,00%	3,80%
6 x 5	47,93	63,50	54,59	54,64	2,70%	18,53%
8 x 2	187,98	188,08	188,02	187,98	0,00%	0,02%
8 x 3	112,98	113,30	113,06	112,98	0,00%	0,08%
8 x 4	78,25	78,25	78,25	78,25	0,00%	0,00%
8 x 5	57,33	62,08	58,56	57,93	0,00%	2,26%
Média Total	**119,68**	**121,61**	**120,43**	**120,34**	**0,17%**	**1,64%**

Tabela B.5: Resultados do GIVMP em todas as instâncias pequenas disponíveis em SOA (2011), com t = 30

Instância	Melhor	Pior	Média	Mediana	RPD_{best}	RPD_{avg}
10 x 2	231,03	231,33	231,08	231,03	0,00%	0,02%
10 x 3	130,08	130,08	130,08	130,08	0,00%	0,00%
10 x 4	97,88	98,20	97,98	98,03	0,00%	0,12%
10 x 5	71,38	72,60	71,60	71,50	0,00%	0,34%
11 x 2	265,13	265,93	265,23	265,13	0,00%	0,04%
11 x 3	156,63	156,63	156,63	156,63	0,00%	0,00%
11 x 4	108,23	108,60	108,33	108,31	0,00%	0,09%
11 x 5	78,53	79,45	78,65	78,58	-0,16%	0,01%
6 x 2	143,35	143,65	143,46	143,35	0,00%	0,07%
6 x 3	88,40	89,15	88,64	88,40	0,00%	0,20%
6 x 4	59,95	70,30	62,77	61,16	0,25%	5,49%
6 x 5	46,80	68,95	53,73	51,10	0,22%	16,26%
8 x 2	187,98	187,98	187,98	187,98	0,00%	0,00%
8 x 3	112,98	113,28	113,03	112,98	0,00%	0,04%
8 x 4	78,25	79,18	78,28	78,25	0,00%	0,03%
8 x 5	57,33	61,73	58,34	57,75	0,00%	1,97%
Média Total	**119,62**	**122,31**	**120,36**	**120,01**	**0,02%**	**1,54%**

Tabela B.6: Resultados do GIVMP em todas as instâncias pequenas disponíveis em SOA (2011), com t = 50

Instância	Melhor	Pior	Média	Mediana	RPD_{best}	RPD_{avg}
10 x 2	231,03	231,03	231,03	231,03	0,00%	0,00%
10 x 3	130,08	130,08	130,08	130,08	0,00%	0,00%
10 x 4	97,88	98,10	97,94	97,90	0,00%	0,08%
10 x 5	71,38	72,45	71,52	71,50	0,00%	0,23%
11 x 2	265,13	265,68	265,25	265,13	0,00%	0,04%
11 x 3	156,63	156,63	156,63	156,63	0,00%	0,00%
11 x 4	108,23	108,53	108,33	108,30	0,00%	0,08%
11 x 5	78,53	79,50	78,60	78,53	-0,16%	-0,05%
6 x 2	143,35	143,38	143,35	143,35	0,00%	0,00%
6 x 3	88,40	89,53	88,51	88,40	0,00%	0,11%
6 x 4	59,75	73,33	62,51	61,48	0,00%	4,76%
6 x 5	46,90	71,38	53,78	51,15	0,64%	16,35%
8 x 2	187,98	187,98	187,98	187,98	0,00%	0,00%
8 x 3	112,98	113,28	113,06	112,98	0,00%	0,06%
8 x 4	78,25	78,30	78,25	78,25	0,00%	0,00%
8 x 5	57,33	62,53	58,39	57,65	0,00%	2,09%
Média Total	**119,61**	**122,60**	**120,33**	**120,02**	**0,03%**	**1,48%**