Universidade Federal dos Vales do Jequitinhonha e Mucuri

Faculdade de Ciências Exatas e Tecnológicas

Departamento de Computação

Curso de Sistemas de Informação

Algoritmos para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência

Luciano Geraldo Silva

Diamantina

2017

Universidade Federal dos Vales do Jequitinhonha e Mucuri

Faculdade de Ciências Exatas e Tecnológicas

Departamento de Computação

Curso de Sistemas de Informação

Algoritmos para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência

Luciano Geraldo Silva

Monografia apresentada ao Curso de Sistemas de Informação do Departamento de Computação da Universidade Federal dos Vales do Jequitinhonha e Mucuri como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Marcelo Ferreira Rego

Diamantina

2017

Algoritmos para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência

Luciano Geraldo Silva

Monografia apresentada ao Curso de Sistemas de Informação do Departamento de Computação da Universidade Federal dos Vales do Jequitinhonha e Mucuri como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em 01 de Setembro de 2017

Marcelo Ferreira Rego

Prof. Me. Marcelo Ferreira Rego – Orientador
Universidade Federal dos Vales do Jequitinhonha e Mucuri

Prof. Dr. Alessandro Vivas Andrade
Universidade Federal dos Vales do Jequitinhonha e Mucuri

Luciana Pereira de Assis

Profª. Drª. Luciana Pereira de Assis
Universidade Federal dos Vales do Jequitinhonha e Mucuri

Diamantina
2017

Este trabalho é dedicado à minha mãe, Petrina Braz Silva.

Agradecimentos

Agradeço primeiramente à Jeová Deus por ter me dado paciência, força, perseverança, determinação e tudo que precisei para chegar até aqui e também por todas as outras coisas que Ele tem me dado ao longo dos anos, pois "toda boa dádiva e todo presente perfeito vem de Jeová." (Tiago 1:17).

Agradeço aos meus pais que me apoiaram, e principalmente a minha mãe que sempre esteve do meu lado nos momentos bons e também nos momentos ruins quando tudo parecia perdido. Agradeço à ela por todas as palavras de conforto, por sempre acreditar no meu potencial, por me apoiar e ajudar em tudo. Sem a ajuda e apoio dela eu nunca teria conseguido chegar tão longe, e a cada dia eu agradeço muito a Jeová por ter ela como mãe! Não tenho palavras para descrever o quão grato sou por ter uma mãe no pleno sentido da palavra, uma mãe que sempre lutou e nunca desistiu, que nunca deixou faltar nada e sempre esteve me esperando de braços abertos, e pra essa pessoa que eu tanto admiro e amo vai o meu mais profundo agradecimento!

Agradeço especialmente ao professor Marcelo que gentilmente aceitou ser o meu orientador e ao longo dessa minha jornada me ensinou e ajudou muitíssimo.

Agradeço à professora Luciana Assis e o professor Alessandro Vivas por disponibilizarem e dedicarem seus preciosos tempos avaliando meu trabalho.

Agradeço à todos os professores que contribuíram para a minha formação.

Agradeço à todo o conhecimento e experiências que obtive por meio da Next Step, empresa júnior de Sistemas de Informação da UFVJM e também à todos os seus membros.

Agradeço à todos os amigos que consegui ao longo dessa jornada de graduação, principalmente aos que me incluíram no seu "clã" (Brian Azevedo, Breno Caldeira, Cristian Fernandes, Luiz Felipe Evaristo, Matheus Guedes, Marcus Paulo Pereira, Rafael Pelli, Pedro Henrique, Réggis Mota e Thomaz Souto), em que tenho muito orgulho de fazer parte.

Agradeço especialmente aos meus dois grandes amigos, Viviane Guimarães e Paulo Henrique Vieira, que são muito importantes e especiais na minha vida e sempre ficaram do meu lado, me apoiaram, foram bons ouvintes e não me deixaram desanimar nem desistir.

Agradeço toda minha família pelo apoio e carinho que sempre demonstraram.

"E parem de se amoldar a este mundo,
mas sejam transformados, renovando a sua mente,
a fim de comprovar por si mesmos a boa,
aceitável e perfeita vontade de Deus."
(Tradução do Novo Mundo da Bíblia Sagrada, Romanos 12: 2)

Resumo

Neste trabalho é estudado o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência (UPMSP – *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*). Nesse problema, um conjunto de tarefas devem ser processadas por um conjunto de máquinas independentes, as tarefas são associadas a um tempo de processamento que é diferente para cada tarefa e depende da máquina a qual ela foi alocada. Além disso, antes de executar cada tarefa, é necessário um tempo para preparar a máquina para executar aquela tarefa, esse é o tempo de preparação, ou tempo de *setup*, que depende da sequência em que as tarefas serão executadas e também depende da máquina utilizada. O objetivo considerado nesse problema é a minimização do maior tempo de conclusão do conjunto de máquinas, também conhecido como *makespan*. Esse problema é muito relevante tanto em sentido prático, pois é muito encontrado no âmbito industrial, como no sentido teórico por pertencer a classe *NP-Difícil*. Com o foco de encontrar boas soluções para o UPMSP foram implementados e avaliados cinco algoritmos. O primeiro é um método exato que foi utilizada a API de C++ do resolvedor *Gurobi* em que o tempo de execução foi limitado em uma hora. O segundo é a metaheurística *Variable Neighborhood Search* (VNS) que utiliza o conceito de estruturas de vizinhança para explorar o espaço de busca, procurar as melhores soluções e escapar de ótimos locais. O terceiro algoritmo é um Algoritmo Genético (*Genetic Algorithm* – GA) que é um método de otimização baseado no processo de evolução natural que foi adaptado para o UPMSP. O quarto e quinto são os algoritmos *Fix-and-Optimize* (F&O) e *Relax-and-Fix* (R&F), que são heurísticas baseadas na formulação matemática do problema. Neste trabalho, para as heurísticas matemáticas, foi adotado o conceito de particionamento do conjunto de variáveis do problema e para resolver os subproblemas particionados é aplicado o resolvedor *Gurobi* com tempo de execução máximo de uma hora. Os resultados obtidos pelos algoritmos foram comparados entre si e a heurística matemática F&O, utilizando a metaheurística VNS para gerar a solução inicial, apresentou os melhores resultados dentre os algoritmos implementados e em algumas instâncias encontrou resultados melhores que os melhores resultados conhecidos disponíveis por (VALLADA; RUIZ, 2011) em (SOA-ITI, 2013).

Palavras-chaves: sequenciamento em máquinas paralelas; UPMSP; *makespan*; metaheurísticas; heurísticas matemáticas; estruturas de vizinhança; *Variable Neighborhood Search*; *Genetic Algorithm*; *Fix-and-Optimize*; *Relax-and-Fix*; *Gurobi*.

Abstract

In this work has been studied the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times, or just UPMSP. In this problem, a set of jobs must be processed by a set of independent machines, the jobs are associated with a processing time that is different for each job and depends on the machine that it was allocated. Before the execution of each one of the jobs, a time is required to set-up the machine to process the job, it is called setup time and depends on the sequence in which the job will be executed on the machine used. The objective considered in this problem is the minimization of the maximum completion time of the schedule, also known as *makespan*. This problem is very relevant both in practical sense, since it is very found in the industrial scope, as well as in theoretical sense because it belongs to the *NP-Hard* class. With the focus of finding good solutions for the UPMSP, five algorithms were implemented and evaluated. The first one is an exact method that has been used the C++ API of the *Gurobi* solver, however the execution time has been limited by one hour. The second one is the *Variable Neighborhood Search* (VNS) metaheuristic that uses the concept of neighborhood structures to explore the search space, find better solutions, and escape from optimal local. The third algorithm is a Genetic Algorithm (GA) which is an optimization method based on the natural evolution process that has been adapted for UPMSP. The fourth and fifth ones are the *Fix-and-Optimize* (F&O) and *Relax-and-Fix* (R&F) heuristics, which are heuristics based on the mathematical formulation of the UPMSP problem In this work, for the mathematical heuristics, the concept of partitioning the problem variables set was adopted and to solve the partitioned subproblems, the *Gurobi* solver was applied at a maximum execution time of one hour. The results obtained by the algorithms were compared to each other and the F&O heuristic with the VNS metaheuristic genereting the initial solution presented the best results among the algorithms implemented and in some instances found better results than the best known results available by (VALLADA; RUIZ, 2011) at (SOA-ITI, 2013).

Key-words: unrelated parallel machines scheduling; UPMSP; makespan; metaheuristic; mathematical heuristics; neighborhood structures; Variable Neighborhood Search; Genetic Algorithm; Fix-and-Optimize; Relax-and-Fix; *Gurobi*.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

F&O	*Fix-and-Optimize*
GA	*Genetic Algorithm* (Algoritmo Genético)
R&F	*Relax-and-Fix*
VNS	*Variable Neighborhood Search*
UPMSP	*Unrelated Parallel Machine Scheduling Problem With Sequence Dependent Setup Times* (Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência)

Lista de Algoritmos

Sumário

1 Introdução

Este trabalho trata do problema de sequenciamento de tarefas em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência (UPMSP – *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*). O problema UPMSP faz parte de um grupo de problemas de sequenciamento que podem ser encontrados em diversas áreas da ciência e também no contexto industrial. Esses problemas de sequenciamento surgiram na literatura no início da década de cinquenta (PINEDO, 2008), e desde então muitos grupos de pesquisa têm trabalhado neles, pois possuem importância tanto na parte prática, por estarem presentes em aplicações reais, tanto quanto na parte teórica, por pertencerem a uma classe de problemas de difícil solução. No UPMSP, existe um conjunto de tarefas que devem ser alocadas num conjunto de máquinas independentes, de tal forma à minimizar o tempo total de execução de todas as tarefas (*makespan*). Assim, o objetivo considerado no problema é encontrar a melhor sequência de alocação das tarefas nas máquinas com a finalidade de minimizar o tempo total de execução.

Ao longo desse trabalho serão abordados cinco algoritmos para a solução do problema UPMSP. Os algoritmos foram executados e os resultados obtidos por cada um deles foi comparado a fim de verificar o método mais eficiente. Os algoritmos foram testados e comparados na resolução das instâncias do problema UPMSP fornecidas por (VALLADA; RUIZ, 2011) em (SOA-ITI, 2013). Foram implementados os seguintes algoritmos: VNS que é uma metaheurística que utiliza o conceito de estruturas de vizinhança para encontrar soluções melhores e fugir de ótimos locais; foi utilizado o resolvedor *Gurobi* em que o tempo de execução máximo foi limitado em uma hora. O terceiro é um Algoritmo Genético (GA) que é um método de otimização baseado no processo de evolução natural que foi adaptado para o problema UPMSP. O quarto e quinto são heurísticas baseadas na formulação matemática do problema UPMSP, nomeadas *Relax-and-Fix* (R&F) e *Fix-and-Optimize* (F&O), que utilizam o conceito de particionamento do conjunto de variáveis do problema e resolver os subproblemas particionados através do resolvedor *Gurobi* com tempo de execução limitado em uma hora.

1.1 Motivação

O estudo do problema de sequenciamento de tarefas, segundo (PINEDO, 2008) tem importância prática e teórica. A relevância prática para este problema está no fato dele pertencer à classe *NP-Difícil*, por isso, não é uma tarefa trivial encontrar a solução ótima para ele. Além disso, o desenvolvimento de métodos eficientes capazes de gerar boas soluções em tempo razoável para essa classe de problemas é importante para o desenvolvimento de áreas do conhecimento como Ciência da Computação, Pesquisa Operacional dentre outas.

Do ponto de vista prático, a solução do problema de sequenciamento de diversas tarefas em máquinas não relacionadas é importante, pois ele é um problema do mundo real, encontrado em algumas áreas da indústria, como por exemplo: eletrônica (KIM et al., 2002), siderúrgica (TANG; WANG, 2009), têxtil (GENDREAU; LAPORTE; GUIMARÃES, 2001), e outras (RAVETTI, 2007). A principal motivação é melhorar o uso de recursos no processo de fabricação e assim reduzir custos e aumentar os lucros.

1.2 Objetivos

Nesta seção são apresentados os objetivos gerais e específicos dessa monografia.

1.2.1 Objetivo Geral

O objetivo principal desse trabalho é implementar algumas heurísticas baseadas na formulação matemática do problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência (UPMSP), comparando seus resultados com as principais metaheurísticas encontradas na literatura e também com a aplicação de um método exato.

1.2.2 Objetivos Específicos

Para que seja possível alcançar o objetivo geral é preciso realizar os objetivos específicos que se segue:

- Implementar o modelo matemático para o problema UPMSP (VALLADA; RUIZ, 2011) no resolvedor comercial *Gurobi*;
- A partir do modelo matemático implementar as heurísticas matemáticas *Fix-and-Optimize* (F&O) e *Relax-and-Fix* (R&F);
- Implementar metaheurística *Variable Neighborhood Search* (VNS) para o UPMSP;
- Implementar *Genetic Algorithm* (GA) utilizando os mesmos parâmetros usados por (VALLADA; RUIZ, 2011) no GA que foi proposto no trabalho desses autores;
- Avaliar os resultados do uso das metaheurísticas e das heurísticas baseadas na formulação matemática;
- Realizar comparações entre os resultados obtidos pelas metaheurísticas implementadas com as heurísticas baseadas na formulação matemática.

1.3 Organização da Monografia

Os próximos capítulos desta monografia estão organizados como se segue.

No Capítulo 2 a fundamentação teórica que serviu como base para esse trabalho, bem como a descrição e formulação matemática do problema UPMSP são descritos. A metodologia aplicada para o desenvolvimento dessa monografia está no Capítulo 3, os resultados computacionais obtidos com a aplicação das técnicas apresentadas e a descrição dos problemas-teste usados para testar e avaliar as mesmas, são apresentados no Capítulo 4. A conclusão deste trabalho juntamente com as propostas de possíveis trabalhos futuros são realizadas no Capítulo 5.

2 Fundamentação Teórica

Neste Capítulo, a seção 2.1 descreve e exemplifica o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência, e também é apresentada a formulação matemática. Os trabalhos que foram usados como base para essa monografia são apresentados na seção 2.2.

2.1 Descrição do Problema

2.1.1 Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência

O problema de sequenciamento de tarefas em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência (UPMSP), possui um conjunto $N = \{1, \ldots, n\}$ de tarefas e um conjunto $M = \{1, \ldots, , m\}$ de máquinas, todas as N tarefas devem ser processadas por uma das máquinas. Esse problema de sequenciamento possui as seguintes propriedades:

(i) Cada tarefa j, $j \in N$, deve ser processada uma única vez e por apenas uma máquina i, $i \in M$;

(ii) Cada tarefa j possui um tempo de processamento P_{ij}, se a tarefa for processada pela máquina i, esse tempo depende da máquina em que a tarefa será alocada;

(iii) Entre o processamento de uma tarefa j e outra tarefa k, $j, k \in N$, existe um tempo de preparação S_{ijk}, também chamado de tempo de *setup*, que representa o tempo que a máquina i necessita para ser configurada antes de processar a tarefa k, logo após de a tarefa j ter sido processada. Esse tempo depende da sequência das tarefas na máquina e da máquina.

(iv) O tempo de preparação para a primeira tarefa, é representado por S_{i0k} em que 0 denota uma tarefa fictícia que precede a tarefa k que será executada na máquina i. O tempo de processamento e o tempo de preparação para essa tarefa fictícia são iguais a zero.

Segundo (PINEDO, 2008) o problema de sequenciamento de máquinas paralelas (PMSP) com máquinas idênticas e sem tempos de preparação ($Pm \mid\mid C_{max}$) pertence à classe *NP-Difícil*. Como o UPMSP é uma generalização do PMSP que acrescenta tempos de preparação ao problema, então esse problema pertence à classe *NP-Difícil*, quando ($Pm \mid prec \mid C_{max}$) com $2 \leq m < n$.

Com objetivo de facilitar o entendimento do problema, foi retirado o seguinte exemplo de (SILVA, 2014), em que é utilizado os dados de uma instância de (VALLADA; RUIZ, 2011). Um cenário hipotético é descrito a seguir:

A empresa fictícia *Fios & Tecidos*, produz e vende seis tipos diferentes de tecidos, para produzir esses tecidos a *Fios & Tecidos* utiliza de duas máquinas que podem trabalhar simultaneamente e produzir qualquer um dos 6 tipos de tecidos, porém cada máquina é limitada a produzir apenas um tipo de tecido por vez. Para produzir cada tecido é necessário um tempo de configuração da máquina para aquele tipo específico de tecido, e após essa configuração prévia, a máquina precisa de um determinado tempo para processar e assim produzir o tecido. Por possuírem modelos diferentes de máquinas, o tempo de configuração da máquina e também o tempo gasto para se produzir o tecido varia de uma máquina para a outra, deste modo, produzir um tipo de tecido em uma máquina pode demorar mais tempo que produzir o mesmo tecido na outra máquina, por esse motivo a *Fios & Tecidos* está querendo saber qual é a sequência de tecidos que cada máquina deve processar, de modo a resultar num tempo total de produção menor, pois a empresa possui uma demanda diária de produção dos 6 tecidos e deseja usar da melhor forma o tempo que as máquinas gastam para produzir todos esses tecidos. Dessa forma, quanto menos tempo as máquinas gastarem para terminar de produzir todos os tecidos da demanda diária, menor é o gasto de produção e a empresa conseguirá um lucro maior, usando seus recursos de maneira inteligente em que se obtém os resultados esperados com lucros maximizados e esforços reduzidos.

A Tabela 1 mostra o tempo que cada uma das máquinas da *Fios & Tecidos* precisa para produzir cada tipo de tecido, as linhas representam as máquinas (M_1, M_2) e as colunas cada tipo de tecido que deve ser produzido ($1, 2, 3, 4, 5, 6$), e por os modelos das máquinas serem diferentes, cada máquina gasta um tempo diferente para produzir cada tecido.

	1	2	3	4	5	6
M_1	8	42	72	45	4	25
M_2	77	4	52	73	10	53

Tabela 1: Tempo de processamento para produção de cada tipo de tecido em cada máquina da *Fios & Tecidos* – (SILVA, 2014)

As máquinas necessitam de um tempo para serem configuradas antes de iniciar a produção de cada tecido, e esse tempo, que é demandado antes de iniciar o processamento do tecido na máquina, é denominado tempo de preparação, ou tempo de *setup*. Ele varia dependendo da máquina que vai produzir o tecido e também de acordo com o tecido que foi produzido anteriormente naquela mesma máquina. Assim, a sequência de tecidos que serão produzidos em determinada máquina interferem diretamente no tempo total de produção gasto por ela. A Tabela 2 mostra os tempos de preparação necessários para que cada máquina esteja pronta para produzir um tecido qualquer logo após ter sido produzido naquela mesma máquina outro

tecido. Por exemplo, se for produzir na máquina M_1 o tecido 3 logo após ter sido produzido o tecido 2 na mesma máquina, o tempo de preparação que a máquina vai necessitar é de 3 unidades de tempo, porém se inverter a ordem de produção dos tecidos, o tempo necessário seria de 29 unidades de tempo, mas ao produzir os mesmos dois tecidos na máquina M_2 ao invés da M_1 o tempo de preparação necessário seria de 46 unidades de tempo. A diagonal principal das matrizes possuem o valor zero porque um tecido não pode ser precedido por ele mesmo. E quando um tecido é o primeiro a ser produzido na máquina, o seu tempo de preparação é igual a zero, pois a máquina não necessita de configuração na produção do primeiro tecido, no entanto para produzir os tecidos subsequentes é preciso uma configuração prévia da máquina antes que seja possível produzir cada tecido da lista de tecidos que foram alocados para serem produzidos nessa máquina.

M_1	1	2	3	4	5	6
1	0	40	31	13	41	24
2	23	0	3	33	33	21
3	12	29	0	18	5	42
4	29	42	2	0	33	33
5	42	18	41	40	0	46
6	4	13	26	37	39	0

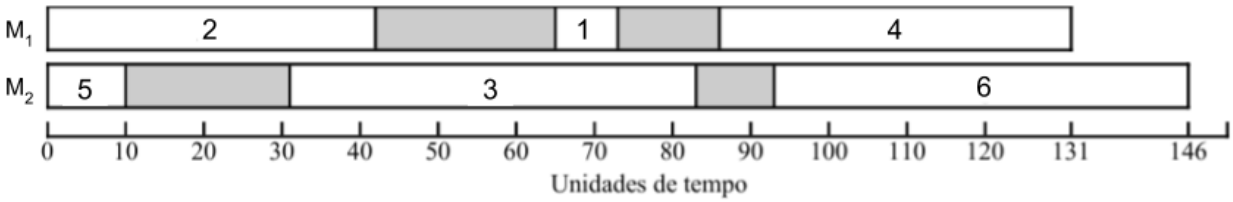
M_2	1	2	3	4	5	6
1	0	23	19	9	44	24
2	7	0	46	31	29	5
3	22	2	0	35	7	10
4	5	18	31	0	22	29
5	15	35	21	5	0	35
6	48	11	40	21	35	0

Tabela 2: Tempo de *setup* entre a produção de dois tecidos para cada máquina da *Fios & Tecidos* – (SILVA, 2014)

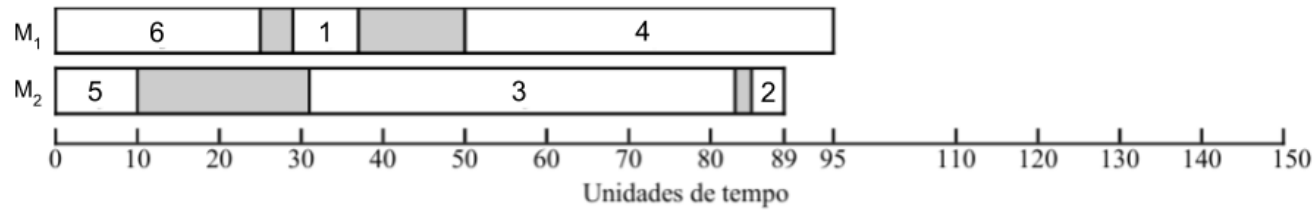
A Figura 1a ilustra uma ordenação qualquer que a *Fios & Tecidos* utiliza para produzir os seus tecidos nas suas máquinas. Nessa solução adotada, a máquina 1 vai produzir os tecidos $2, 1$ e 4, nessa ordem, e os tecidos $5, 3$ e 6 são produzidos pela máquina 2 também nessa ordem. Na figura, os retângulos brancos representam os tempos de processamento que cada máquina gasta para produzir os tecidos e os retângulos cinzas representam os tempos de preparação, que cada máquina necessita para produzir dois tecidos consecutivos.

Porém a solução ótima, que faria a *Fios & Tecidos* obter o maior lucro é ilustrada na Figura 1b. É possível notar que existe uma redução considerável do tempo total de produção que será necessário para produzir todos os tecidos, esse tempo total muda de 146 unidades de tempo na primeira solução para 95 unidades. Apenas por fazer uma ordenação melhor da sequência de tecidos que serão produzidos e escolher qual é a melhor máquina que deve produzir cada sequência, é possível otimizar o tempo de produção e economizar 51 unidades de tempo, resultando em menor gasto para a empresa *Fios & Tecidos* e usando de modo eficaz seus recursos evitando gastos desnecessários.

Note que nesse exemplo fictício, o problema de UPMSP foi descrito em uma possível situação do mundo real que utiliza máquinas paralelas no planejamento da produção. Em



(a) Solução qualquer



(b) Solução ótima

Figura 1: Soluções para produção de tecidos na *Fios & Tecidos* – (SILVA, 2014)

situações semelhantes, deseja-se a melhor sequência de execução das tarefas nas máquinas com o objetivo de obter economia de tempo e recursos para a empresa.

2.1.2 Formulação Matemática

O Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempos de Preparação Dependentes da Sequência, UPMSP (do inglês, Unrelated Parallel Machine Scheduling Problem With Sequence Dependent Setup Times) é modelado por (VALLADA; RUIZ, 2011) através de um modelo de Programação Inteira Mista, MIP (do inglês, Mixed Integer Programming). E os dados do problema são definidos da seguinte forma:

M : conjunto de máquinas paralelas não relacionadas.

N : conjunto de tarefas que serão processadas nas máquinas.

p_{ij} : tempo de processamento da tarefa j na máquina i.

S_{ijk} : tempo de preparação que a máquina i necessita quando a tarefa k for processada logo após o processamento da tarefa j.

Com as variáveis de decisão:

$$X_{ijk} : \begin{cases} 1, & \text{se a tarefa } j \text{ precede a tarefa } k \text{ na máquina } i \\ 0, & \text{caso contrário} \end{cases}$$

C_{ij} : Tempo para completar a tarefa j na máquina i

C_{max} : makespan, é o maior tempo de conclusão das máquinas que processam as tarefas

programadas

O objetivo do problema é encontrar o menor tempo possível para executar todas as tarefas propostas utilizando de todas as máquinas disponíveis e respeitando as restrições definidas para cada tarefa em cada máquina, sendo assim, a função objetivo minimiza o maior tempo de conclusão das máquinas, ou *makespan*, e é dada por:

$$min \quad C_{max} \tag{2.1}$$

As restrições para o modelo são:

$$\sum_{i \in M} \sum_{\substack{j \in \{0\} \cup \{N\} \\ j \neq k}} X_{ijk} = 1, \qquad \forall k \in N, \tag{2.2}$$

$$\sum_{i \in M} \sum_{\substack{k \in N \\ j \neq k}} X_{ijk} \leq 1, \qquad \forall j \in N, \tag{2.3}$$

$$\sum_{k \in N} X_{i0k} \leq 1, \qquad \forall i \in M, \tag{2.4}$$

$$\sum_{\substack{h \in \{0\} \cup \{N\} \\ h \neq k, h \neq j}} X_{ihj} \geq X_{ijk}, \qquad \forall j, \quad k \in N, \quad j \neq k, \quad \forall i \in M, \tag{2.5}$$

$$C_{ik} + V(1 - X_{ijk}) \geq C_{ij} + S_{ijk} + p_{ik}, \qquad \forall j \in \{0\} \cup \{N\}, \quad \forall k \in N, \quad j \neq k, \quad \forall i \in M, \tag{2.6}$$

$$C_{i0} = 0, \qquad \forall i \in M, \tag{2.7}$$

$$C_{ij} \geq 0, \qquad \forall j \in N, \quad \forall i \in M, \tag{2.8}$$

$$C_{max} \geq C_{ij}, \qquad \forall j \in N, \quad \forall i \in M, \tag{2.9}$$

$$X_{ijk} \in \{0, 1\}, \qquad \forall j \in \{0\} \cup \{N\}, \quad \forall k \in N, \quad j \neq k, \quad \forall i \in M. \tag{2.10}$$

Esse modelo de acordo com (SILVA, 2014) inclui tarefas fictícias, que possuem tempos de preparação e processamentos iguais a zero, com objetivo de que essas tarefas precedam a primeira tarefa do problema, para atender as equações (2.7) e (2.10). Para garantir que o tempo de preparação seja zero em todas as tarefas fictícias adicionadas ao problema, (SILVA, 2014) acrescenta ao modelo matemático a seguinte restrição:

$$S_{i0k} = 0, \qquad \forall k \in N, \quad \forall i \in M. \tag{2.11}$$

A restrição (2.2) assegura que cada tarefa é designada para somente uma máquina, do conjunto de máquinas disponíveis, e possui somente uma tarefa predecessora. Na restrição (2.3) é limitado em 1 o número máximo de tarefas sucessoras de cada tarefa, por as últimas tarefas não possuírem sucessoras o valor pode ser igual a 0. Igualmente para as tarefas fictícias, o número máximo de tarefas sucessoras, para cada tarefa, é no máximo 1 em cada máquina, conforme a (2.4). Em (2.5) é verificada a distribuição correta das tarefas, assegurando que se uma tarefa j é processada na máquina i, existe uma tarefa h na mesma máquina i que é antecessora de j, por causa dessa restrição as tarefas fictícias são necessárias. A restrição (2.6) é para controlar o tempo de conclusão das tarefas nas máquinas. Assim se uma tarefa k é designada para uma máquina i após a tarefa j ($X_{ijk} = 1$), seu tempo de conclusão C_{ik} deve ser maior que o tempo de conclusão de C_{ij} acrescido do tempo de preparação entre j e k e o tempo de processamento de k. Se $X_{ijk} = 0$, a constante V, um valor qualquer arbitrariamente grande, tornará a restrição (2.6) redundante. Para as tarefas fictícias, (2.7) garante que o tempo de conclusão de todas essas tarefas seja igual a 0. Em tarefas regulares, não fictícias, a restrição (2.8) define que o tempo de conclusão deve ser maior ou igual a 0. O tempo de conclusão máximo, $makespan$, é definido na restrição (2.9) e a restrição (2.10) define as variáveis binárias do problema.

2.2 Trabalhos Relacionados

Nessa seção são descritos alguns trabalho que abordam o UPMSP e propõem metaheurísticas para encontrar boas soluções para esse problema.

Baseado em um caso real de problema de sequenciamento em uma empresa brasileira da cidade de Belo Horizonte, (RAVETTI, 2007) implementa as metaheurísticas *Greed Randomized Adaptive Search Procedure* (GRASP) e *Variable Neighborhood Search* (VNS). Com objetivo de minimizar o tempo de conclusão de todas as tarefas mais a soma dos seus atrasos ponderados. O problema abordado por pelo autor se difere do que será feito nesse trabalho, pois ele impõe datas de entrega para cada tarefa. Os algoritmos mostraram ser eficientes, gerando bons resultados em instâncias grandes e soluções ótimas em instâncias pequenas, porém o algoritmo VNS mostrou ser o mais eficiente em instâncias com 60 tarefas ou mais.

(VALLADA; RUIZ, 2011) propõem dois Algoritmos Genéticos, GA1 e GA2, para o UPMSP, que se diferem entre si pelos parâmetros adotados para tamanho de população,

probabilidade de cruzamento, probabilidade de mutação, probabilidade de aplicação da busca local e pressão seletiva. Também utilizaram o resolvedor comercial CPLEX[1] da IBM para encontrar soluções ótimas em instâncias pequenas do problema. A minimização do tempo de conclusão de todas as tarefas, também chamado de *makespan*, é o objetivo do trabalho. As instâncias para o problema que utilizaram, que também serão utilizadas nesse trabalho, foram criadas pelos dois autores e será explicada em mais detalhes na seção 4.1. O algoritmo denotado por GA2, com os 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, foi o que obteve os melhores resultados para as instâncias grandes e pequenas do problema.

(HADDAD, 2012) apresentou três algoritmos heurísticos híbridos com objetivo de minimizar o *makespan* nesse problema de sequenciamento. O autor aplicou e avaliou os algoritmos desenvolvidos em dois conjuntos de instâncias, as criadas por (VALLADA; RUIZ, 2011) e também nas instâncias de (RABADI; MORAGA; AL-SALEM, 2006). Os algoritmos propostos são o IVP que combina as heurísticas *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Path Relinking* (PR); o GIVP que se diferencia do primeiro por utilizar *Greed Randomized Adaptive Search Procedure* (GRASP) para gerar a solução inicial e o GIVMP que combina a fase de construção *Greed Randomized Adaptive Search Procedure* (GRASP), os procedimentos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e na fase de *Path Relinking* usa *Mixed Relinking* ao invés de *Backward Relinking* e um módulo de busca local feito por um resolvedor de programação inteira mista, o resolvedor comercial CPLEX[1]. O autor concluiu que o algoritmo GIVMP se mostrou melhor que os outros dois e os resultados gerados pelo GIVMP são absolutamente superiores aos algoritmos genéticos propostos por (VALLADA; RUIZ, 2011), esse algoritmo apenas não ganhou do algoritmo MVND proposto por (FLESZAR; CHARALAMBOUS; HINDI, 2011) para as instâncias de (RABADI; MORAGA; AL-SALEM, 2006).

(SILVA, 2014) implementou e avaliou quatro metaheurísticas, relativamente simples, que geraram melhores resultados comparados aos resultados obtidos por (VALLADA; RUIZ, 2011). O objetivo do seu trabalho foi a minimização do *makespan*, tempo de conclusão de todas as tarefas, com as seguintes metaheurísticas: *Simulated Annealing* (SA), *Iterated Local Search* (ILS), *Late Acceptance Hill Climbing* (LAHC), *Step Counting Hill Climbing* (SCHC). Para todas as metaheurísticas foi encontrado diferentes configurações de parâmetros que melhoram consideravelmente os resultados das soluções para instâncias grandes. Os resultados gerados pela metaheurística *Simulated Annealing* com os parâmetros *número de iterações em cada temperatura* (SA_{max}) igual a 200.000, *taxa de resfriamento* (α) com valor de $0,99$, *tempo limite* definido em 60 segundos, e a *temperatura inicial* (T_0) é dada por outros dois parâmetros, *mti* multiplicado por *pro*, recebendo respectivamente os valores 1.000 e $0,05$. Com esses parâmetros

[1] www.cplex.com

os resultados encontrados foram melhores que os obtidos por (VALLADA; RUIZ, 2011) e (HADDAD, 2012) para as instâncias grandes criadas por (VALLADA; RUIZ, 2011) disponíveis em (SOA-ITI, 2013).

(STEFANELLO et al., 2015) propõe uma abordagem híbrida de busca local baseado em metaheurística e algoritmos exatos, com objetivo de minimizar o tempo máximo de conclusão da ultima tarefa, o *makespan*. A heurística utiliza um resolvedor comercial, CPLEX[1], para procurar vizinhos em um algoritmo *multi-start*. Para testar seu algoritmo MIP-based, os autores usaram as instâncias propostas por (VALLADA; RUIZ, 2011), os resultados obtidos superaram os resultados dos algoritmos genéticos propostos por (VALLADA; RUIZ, 2011).

3 Metodologia

Neste Capítulo, é realizado a descrição do modo como os algoritmos utilizados nesse trabalho foram implementados.

3.1 Função Objetivo

No problema UPMSP tem-se um conjunto de n tarefas, que devem ser alocadas em um conjunto de m máquinas. Uma solução do problema consiste em definir quais tarefas devem ser executadas em cada máquina, bem como, estabelecer a ordem de execução das tarefas nas respectivas máquinas. Nesse problema, a sequência de execução das tarefas dentro da máquina é importante, pois cada tarefa tem um tempo diferente de preparação (*setup*), dependendo da tarefa que está sendo executada antes dela na mesma máquina.

A avaliação de uma solução para este problema é feita a partir da função objetivo, que consiste no maior tempo de conclusão entre todas as tarefas, considerando todas as máquinas, esse valor também é conhecido como *makespan*. Para o problema UPMSP, o objetivo é minimizar o maior tempo de conclusão, assim quanto menor o valor do *makespan*, melhor é a solução. Esse critério de avaliação é utilizado em todos os algoritmos implementados que são descritos nas seções seguintes.

3.2 Metaheurísticas

Nessa seção as duas metaheurísticas que foram implementadas nesse trabalho são detalhadas. A subseção 3.2.3 explica o algoritmo baseado na metaheurística *Variable Neighborhood Search*, juntamente com as estruturas de vizinhança e o algoritmo de busca local usado por ela. Já a subseção 3.2.4 descreve a implementação do Algoritmo Genético que foi proposto por (VALLADA; RUIZ, 2011) para o problema UPMSP. As soluções do problema são representadas nas metaheurísticas conforme descrito na subseção 3.2.1.

3.2.1 Representação da Solução

Uma solução s qualquer do problema UPMSP no Algoritmo 2 (VNS) e também cada indivíduo no Algoritmo 3 (GA), é representada por um vetor de máquinas, e cada posição desse vetor possui uma lista de tarefas que são alocadas para aquela máquina e devem ser executadas na ordem em que foram inseridas nessa lista. O tamanho do vetor de máquinas é definido pelo número de máquinas, m, do problema, e a lista de tarefas de cada item do vetor possui tamanho variado, o que define o tamanho da lista de tarefas é a quantidade de tarefas que aquela máquina

executará. As tarefas são representadas por números inteiros de 1 a n, em que n é o número total de tarefas do problema.

A Figura 2 mostra um exemplo de uma representação de solução que possui 3 máquinas e 6 tarefas. Nessa representação, o vetor M possui todas as máquinas do problema e em cada posição desse vetor estão alocadas as tarefas que a máquina processará, assim, conforme a figura, a máquina 1 processará as tarefas 6, 1 e 3; a máquina 2 processará a tarefa 4 e a máquina 3 executará as tarefas 2 e 5. Todas as tarefas serão executadas na mesma ordem que foram inseridas na lista e a execução em cada máquina é independente das outras máquinas, ou seja, as tarefas são executadas paralelamente, independentemente da tarefa que outra máquina está executando no momento. Cada máquina tem o seu próprio tempo de conclusão das tarefas alocadas para ela. O maior tempo de conclusão entre as máquinas, será o valor do *makespan* para a solução.

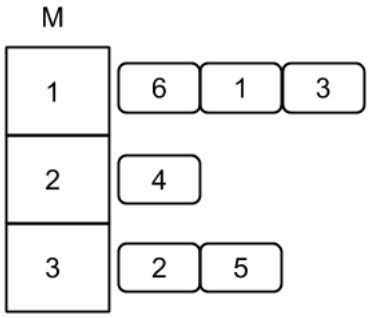


Figura 2: Ilustração da representação de uma solução s do problema UPMSP no VNS e de um indivíduo no GA

3.2.2 Solução Inicial

Neste trabalho, foi implementado uma heurística para construir uma solução inicial para o problema, que também é conhecida como heurística construtiva. Uma heurística construtiva gera soluções partindo de uma solução vazia e iterativamente chega em uma solução, geralmente, viável para o problema, de maneira gulosa, ou seja, visando conseguir a cada passo um melhor valor para a função objetivo. A escolha do melhor elemento a ser inserido na solução em cada iteração da heurística é feita identificando o elemento que traz o melhor benefício ao ser inserido na solução, dentre todos os elementos candidatos. O processo finaliza quando o conjunto de elementos candidatos fica vazio, e consequentemente a solução que inicialmente estava vazia estará completa.

A solução inicial que é fornecida para Algoritmo 2 utiliza um método simples baseado em uma heurística construtiva gulosa para gerar tal solução, em que os tempos de processamento das tarefas em cada máquina são avaliados com a finalidade de alocar a melhor tarefa para cada máquina. O critério de alocação de tarefa por máquina é feito escolhendo qual máquina i, a tarefa j possui o menor tempo de processamento, e ao escolher a máquina mais propicia, a tarefa

j é designada para ser processada nessa máquina i. Esse processo de alocação das tarefas em cada máquina é mostrado no Algoritmo 1.

Algoritmo 1: Construção da Solução Inicial

```
1  para cada (tarefa j ∈ N) faça
2      indice ← 1;
3      processamentoMaquina ← p[1][j];
4      para cada (máquina i ∈ M) faça
5          se (p[i][j] < processamentoMaquina) então
6              indice ← i;
7              processamentoMaquina ← p[i][j];
8          fim
9      fim
10     maquinas[indice] adicionar ao fim da lista a tarefa j;
11 fim
```

A solução inicial também pode ser gerada por meio de uma abordagem aleatória que escolhe aleatoriamente os elementos para adicionar na solução ao invés de usar algum tipo de critério de avaliação para inseri-los. Porém, ao criar soluções iniciais com esse método aleatório, as soluções geradas têm grande probabilidade de serem piores que as geradas usando um método guloso, por esse motivo foi utilizado nesse trabalho apenas o método de heurística construtiva gulosa para a solução inicial do VNS.

3.2.3 Variable Neighborhood Search

De acordo com (HANSEN; MLADENOVIC, 2003), o *Variable Neighborhood Search* (VNS) é uma metaheurística moderna que usa a ideia de explorar sistematicamente mudanças de vizinhanças e com isso, conseguir procurar soluções melhores na vizinhança e também fugir de ótimos locais. Diferentemente de muitas outras metaheurísticas, o VNS é simples e requer poucos ou nenhum parâmetro, no entanto, ele possui grande potencial de encontrar boas soluções no espaço de busca em um período de tempo razoável.

A metaheurística explora o espaço de busca sem seguir uma trajetória, mas explora vizinhanças distantes (HANSEN; MLADENOVIĆ, 2001). O VNS utiliza a estratégia de estruturas de vizinhança para explorar o espaço de busca e procurar soluções que melhorem a solução inicial fornecida. Ao explorar a vizinhança, gerada pela estrutura de vizinhança, o algoritmo faz uma busca local e só aceita soluções vizinhas que melhoram a solução atual. O Algoritmo 2 mostra o pseudo-código do VNS implementado nesse trabalho.

O Algoritmo 2 inicialmente gera uma solução inicial s_0, linha 1, conforme descrito na subseção 3.2.2. Em seguida, s_0 é atribuída à solução corrente s, linha 3. Na sequência, o

algoritmo entra no laço principal, linhas 4-16. Dentro do laço principal, o algoritmo realiza busca local para cada estrutura de vizinhança. Para realizar a busca local, gera-se um vizinho s' a partir da aplicação da estrutura de vizinhança k na solução s, linha 7. Em seguida, ocorre a exploração do espaço de busca com o Método de Primeira Melhora, linha 8. A melhor solução obtida a partir da exploração de s'' é comparada com a melhor solução corrente s, linha 9, caso, s'' seja melhor que s então s'' passa a ser a melhor solução corrente, linha 10 e o método continua percorrendo o espaço de busca a partir da primeira vizinhança, linha 11. Senão, a solução s'' é desconsiderada e o método continua percorrendo o espaço de busca a partir da próxima estrutura de vizinhança, linha 13.

Algoritmo 2: Variable Neighborhood Search – VNS

```
1  Seja s_0 uma solução inicial;
2  Dado o conjunto de estruturas de vizinhança N_k para k = 1, ..., k_max;
3  s ← s_0;
4  enquanto (Critério de parada não for satisfeito) faça
5      k ← 1;
6      enquanto (k ≤ k_max) faça
7          Shake procedure: Gere uma solução vizinha aleatória qualquer s' ∈ N_k(s);
8          Local Search: Aplique um método de busca local fornecendo s' como solução
           inicial do método, e o ótimo local encontrado por essa busca será denotado por
           s'';
9          se (f(s'') < f(s)) então
10             s ← s'';
11             k ← 1;
12         senão
13             k ← k + 1;
14         fim
15     fim
16 fim
```

O algoritmo necessita primeiramente de uma solução inicial e para gerar essa solução inicial, foi utilizado uma heurística construtiva simples, detalhada na subseção 3.2.2.

Na literatura é possível encontrar outras estratégias para se obter uma solução inicial. Como em (SILVA, 2014) o autor utiliza outro método de heurística construtiva para encontrar a solução inicial dos algoritmos desenvolvidos por ele para o problema UPMSP, fazendo uma distribuição sequencial das tarefas nas máquinas. Já (RAVETTI, 2007) utiliza a heurística NEH como solução inicial para o VNS aplicado ao problema de sequenciamento com máquinas paralelas e tempos de preparação dependentes da sequência com datas de entrega para cada tarefa. E (VALLADA; RUIZ, 2011) utiliza para gerar a população inicial dos Algoritmos Genéticos

propostos pelos autores, soluções geradas aleatoriamente e aplicando a heurística *Multiple Insertion*, MI, nessas soluções aleatórias geradas. Considerando que o objetivo do presente trabalho não é a escolha do melhor método para gerar uma solução inicial para o VNS, foi adotado um método simples para a distribuição inicial das tarefas nas máquinas.

Com objetivo de explorar o espaço de busca são utilizadas algumas estruturas de vizinhança que a partir de uma dada solução, geram a vizinhança dessa solução. As estruturas de vizinhança utilizadas no VNS que foi implementado são explicadas em maiores detalhes na subseção 3.2.3.2 e o critério de parada para o algoritmo é definido na subseção 3.2.5.

3.2.3.1 Busca Local

Diferentemente das heurísticas construtivas que partem de uma solução vazia e constroem uma solução factível, as heurísticas de refinamento partem de uma solução inicial factível e têm por finalidade melhorar essa solução aplicando modificações nos seus elementos. Para atingir esse objetivo de melhorar a solução inicialmente fornecida, essas heurísticas utilizam a noção de vizinhança. As vizinhanças são geradas por uma estrutura de vizinhança previamente definida, e por sua vez as estruturas de vizinhança realizam determinados movimentos nos elementos da solução fornecida com a finalidade de gerar novas soluções vizinhas dessa solução dada. Sendo assim escolher boas estruturas de vizinhança para a heurística de refinamento é um fator importante e tem um impacto direto na qualidade das soluções encontradas.

Para explorar o espaço de busca, o *Variable Neighborhood Search* (Algoritmo 2), utiliza uma estrutura de vizinhança em cada iteração do algoritmo e ao obter uma solução vizinha por meio da estrutura de vizinhança, é realizado uma busca local nessa vizinhança, para assim procurar uma solução que melhore a solução corrente. A heurística *First Improvement* foi utilizada como o método de busca local para o VNS implementado nesse trabalho.

A heurística *First Improvement*, ou Método de Primeira Melhora, é uma alternativa a heurística *Best Improvement*, ou Método de Subida/Descida Completa, em que ambas as heurísticas exploram uma vizinhança com objetivo de procurar soluções melhores que a solução atual, mas ao invés de fazer uma busca exaustiva por toda a vizinhança visando encontrar a melhor solução daquela vizinhança, a estratégia do *First Improvement* é interromper a busca na vizinhança no momento em que encontrar uma solução que apresenta uma melhora em relação a solução atual, no pior caso, essa heurística vai ser exatamente igual ao *Best Improvement* que vai vasculhar toda a vizinhança. Com essa estratégia é possível evitar o esforço computacional de vasculhar toda a vizinhança em busca do ótimo local dela e ainda assim encontrar soluções melhores.

Essa heurística de refinamento, apesar de não garantir encontrar o ótimo local da vizinhança, é possível conseguir bons resultados sem o trabalho dispendioso de percorrer toda a vizinhança em busca do ótimo local, e em alguns casos a primeira melhora encontrada pode ser o ótimo local da vizinhança e o fato de a heurística não continuar pesquisando o restante das

soluções faz a busca ser eficiente.

3.2.3.2 Estruturas de Vizinhança

Uma parte crucial da metaheurística *Variable Neighborhood Search* é o uso de boas estruturas de vizinhança para o problema, pois por meio dessas estratégias de estruturas de vizinhança são geradas as vizinhanças das soluções para que seja aplicada a busca local nessa vizinhança, a fim de procurar por uma solução $s^{'}$ melhor que a solução s corrente. Por esse motivo, ao utilizar estruturas de vizinhança eficazes é possível fugir de ótimos locais e também encontrar boas soluções para o problema.

As estruturas de vizinhança tem por objetivo realizar movimentos em uma solução s, que é fornecida como parâmetro, e esses movimentos gerarão novas soluções vizinhas de s. As soluções do problema UPMSP, conforme descrito na subseção 3.2.1, possuem um conjunto de máquinas e cada máquina tem uma sequência de tarefas, deste modo, os movimentos das estruturas de vizinhança realizarão mudanças na ordem das tarefas em cada máquina e também trocas de tarefas entre máquinas, e por meio desses movimentos o valor do *makespan* da solução dada será afetado, gerando assim novas soluções com base na solução s.

Foram utilizados cinco tipos diferentes de estruturas de vizinhança para o Algoritmo 2, essas estruturas de vizinhança foram retiradas de (SILVA, 2014), e a descrição de cada uma delas é feita a seguir:

- *Task Move*: duas máquinas, M_x e M_y, são selecionadas aleatoriamente. Uma tarefa j é escolhida aleatoriamente na máquina M_x, e essa tarefa será removida da máquina M_x e alocada em uma posição aleatória na máquina M_y. Um exemplo desse movimento é ilustrado na Figura 3.

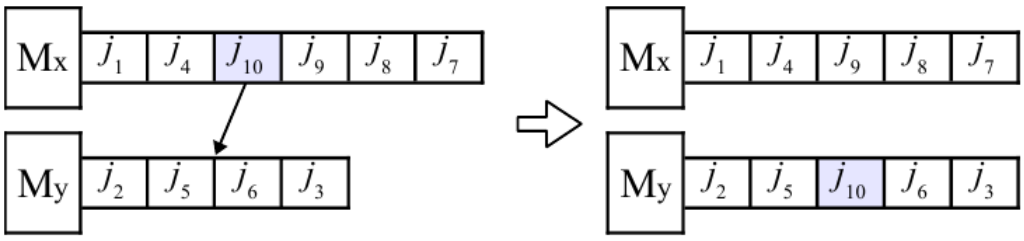


Figura 3: Exemplo de um movimento utilizando a vizinhança *Task Move* – (SILVA, 2014)

- *Shift*: uma máquina, M_x, é selecionada aleatoriamente. Uma tarefa j é selecionada aleatoriamente na máquina M_x, e essa tarefa será removida da posição atual na máquina M_x e realocada em uma outra posição aleatória na mesma máquina M_x. Um exemplo desse movimento é ilustrado na Figura 4.

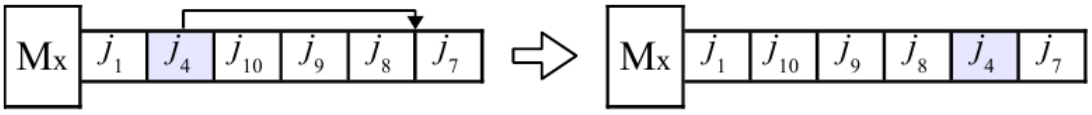


Figura 4: Exemplo de um movimento utilizando a vizinhança *shift* – (SILVA, 2014)

- *Switch*: uma máquina, M_x, é selecionada aleatoriamente. Duas tarefas, j e k são selecionadas também de modo aleatório na máquina M_x. As tarefas j e k vão trocar de posição na máquina M_x, assim a tarefa j será realocada na posição da tarefa k na máquina M_x e a tarefa k será realocada na posição da tarefa j. Um exemplo desse movimento é ilustrado na Figura 5.

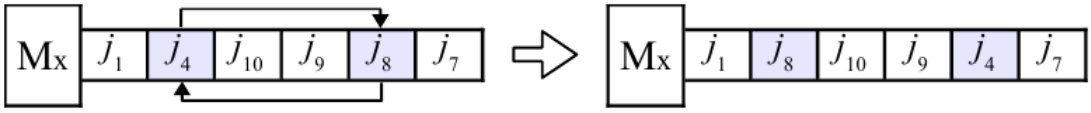


Figura 5: Exemplo de um movimento utilizando a vizinhança *Switch* – (SILVA, 2014)

- *Swap*: duas máquinas, M_x e M_y, são selecionadas aleatoriamente. Uma tarefa j é escolhida da máquina M_x e outra tarefa k é selecionada da máquina M_y, ambas as tarefas são escolhidas aleatoriamente. A tarefa j será removida da máquina M_x e alocada em uma posição aleatória da máquina M_y e a tarefa k será removida da máquina M_y e alocada para uma posição aleatória da máquina M_x. Um exemplo desse movimento é ilustrado na Figura 6.

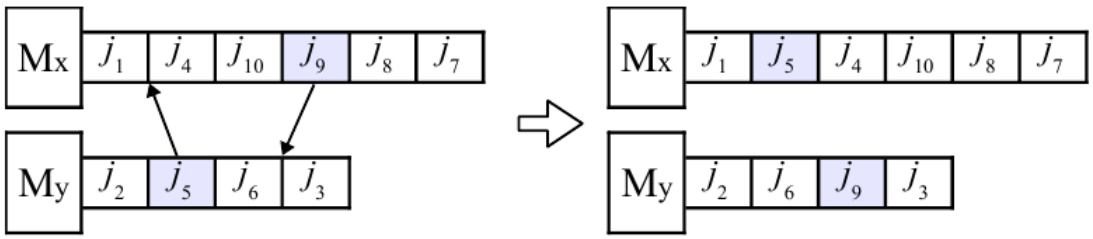


Figura 6: Exemplo de um movimento utilizando a vizinhança *Swap* – (SILVA, 2014)

- *2-realloc*: uma máquina, M_x, é selecionada aleatoriamente. Duas tarefas, j e k são selecionadas de forma aleatória na máquina M_x. A tarefa j será removida da posição atual na máquina M_x e alocada em outra posição aleatória na mesma máquina M_x, o mesmo processo é aplicado a tarefa k. Essa estrutura é parecida com a estrutura *Shift*, porém nela são utilizas duas tarefas e na *Shift* apenas uma. Um exemplo desse movimento é ilustrado na Figura 7.

As máquinas M_x e M_y e as tarefas j e k são sempre escolhidas aleatoriamente. Assim cada uma das estruturas de vizinhança recebe como parâmetro uma solução s e após realizar os

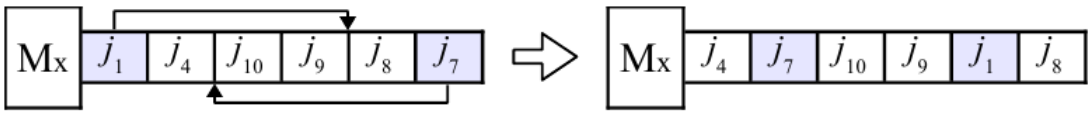


Figura 7: Exemplo de um movimento utilizando a vizinhança *2-realloc* – (SILVA, 2014)

movimentos nas tarefas e máquinas da solução dada será gerado uma nova solução s'. Assim utilizando as estruturas de vizinhança é possível sair de ótimos locais e explorar o espaço de busca.

A sequência que o Algoritmo 2 usará para escolher as estruturas de vizinhança é a mesma sequência em que as estruturas foram descritas, sendo assim o tipo de estrutura inicial $N_{(1)}$ é a estrutura de vizinhança *Task Move* e o tipo de estrutura $N_{(k_{max})}$, em que $k_{max} = 5$, é a estrutura de vizinhança *2-realloc*.

(SILVA, 2014) descreve três critérios para escolher as máquinas que farão parte dos movimentos nas estruturas de vizinhança, em que o primeiro método é denominado Estratégia de Seleção de Máquina Aleatório (*R - Random Machine Selection Strategy*) e nesse critério todas as máquinas têm a mesma probabilidade de serem escolhidas para participarem dos movimentos das estruturas de vizinhança; o segundo critério descrito por ele é a Estratégia de Seleção Baseada na Máquina Makespan (*Mk - Makespan Based Machine Selection Strategy*) em que os movimentos realizados são feitos na máquina que contém o *makespan* da solução, já o terceiro critério é uma Estratégia Mista (*Mi - Mixed Strategy*) que mistura as duas estratégias anteriores em que é selecionado aleatoriamente em cada iteração a estratégia R ou MK. Nesse trabalho é utilizada a estratégia R, em que todas as máquinas possuem a mesma chance de serem selecionadas pela estrutura de vizinhança.

3.2.4 Algoritmo Genético

Segundo (MICHALEWICZ, 1996), os Algoritmos Genéticos (*Genetic Algorithms* – GAs), são algoritmos estocásticos que utilizam conceitos evolutivos da natureza, como os fenômenos de herança genética, para encontrar soluções de um determinado problema. Os GAs adotam também algumas terminologias da genética natural, tais como indivíduos, cromossomos, genes, cruzamento, mutação e população. Os GAs basicamente fazem como a seleção natural que ocorre na natureza, em que os animais mais aptos vivem o suficiente para terem descendentes e os menos aptos morrem. Os GAs seguem esse raciocínio, e por meio de uma população inicial de indivíduos, o algoritmo utiliza técnicas para selecionar indivíduos mais aptos da população, fazendo cruzamentos desses indivíduos mais aptos e misturando seus cromossomos para gerar novos indivíduas que constituirão as gerações futuras, esses descendentes possuem as características dos pais selecionados e por usar o artifício de mutação nesses descendentes é possível conseguir diversidade nas novas gerações. Assim as novas gerações possuem indivíduos melhores, ou mais evoluídos, que as gerações anteriores e como resultado as soluções vão ficando

cada vez melhores à medida que o algoritmo gera novos indivíduos, e os indivíduos considerados ruins vão sendo removidos das novas gerações, resultando em populações melhores.

O Algoritmo 3 mostra o pseudo-código do algoritmo genético apresentado (MICHALEWICZ, 1996). Em que t representa a geração de indivíduos, $P(t)$ é a população de uma geração t, cada indivíduo da população representa uma solução para o problema. Enquanto o critério de parada previamente definido não for atingido, o algoritmo vai gerar novos indivíduos para a nova população, esses indivíduos são gerados a partir da operação de cruzamento que escolhe bons indivíduos por meio de uma operação de seleção, e para gerar uma certa diversidade na população existe a operação de mutação que altera, dada um determinada probabilidade, os novos indivíduos gerados pelos cruzamentos. Por meio desse algoritmo a população inicial vai evoluindo e com essa evolução melhores indivíduos, soluções do problema, são gerados em cada iteração e os indivíduos ruins são descartados da população.

Algoritmo 3: Algoritmo Genético Canônico

```
1 t ← 0;
2 iniciar P(t);
3 avaliar P(t);
4 enquanto (Critério de parada não for satisfeito) faça
5 |   t ← t + 1;
6 |   selecione P(t) a partir de P(t − 1);
7 |   alterar P(t);
8 |   avaliar P(t);
9 fim
```

(VALLADA; RUIZ, 2011) apresentam quatro variações para o Algoritmo Genético proposto no trabalho deles, e para todas as variações, os autores definiram os mesmos parâmetros para tamanho da população, quantidade de indivíduos presentes no processo de seleção de indivíduos mais aptos e as probabilidades de cruzamento e mutação, esses valores definidos por eles estão descritos na Tabela 3. Para maior detalhamento sobre as diferenças entre cada uma das variações do Algoritmo Genético proposto por eles veja em (VALLADA; RUIZ, 2011) na subseção 4.4.

Paramêtro	Valor
Tamanho da População (P_{size})	50
Porcentagem de Indivíduos no Torneio ($pressure$)	30
Probabilidade de Cruzamento (P_c)	0.5
Probabilidade de Mutação (P_c)	0.2

Tabela 3: Valores adotados para os parâmetros das quatro variações para o GA proposto por (VALLADA; RUIZ, 2011)

Após a avaliação das quatro variações foi escolhida a variação denotada como GAStd4,

pois ela mostrou ser a melhor das variações, e os autores fizeram duas calibragens dos parâmetros para essa variação em busca de melhorar os resultados obtidos, na segunda calibragem foi encontrado os melhores valores para os parâmetros dessa variação GAStd4, esses valores estão na Tabela 4. O algoritmo genético implementado nesse trabalho adotou também os mesmos parâmetros descritos na Tabela 4 e também foi utilizado os métodos da variação GAStd4 descrita como sendo a melhor entre as propostas por (VALLADA; RUIZ, 2011).

Paramêtro	Valor
Tamanho da População (P_{size})	80
Porcentagem de Indivíduos no Torneio ($pressure$)	10
Probabilidade de Cruzamento (P_c)	0.5
Probabilidade de Mutação (P_c)	0.5

Tabela 4: Valores dos parâmetros da variação GAStd4 do GA proposto por (VALLADA; RUIZ, 2011) após a calibragem

3.2.4.1 População Inicial e Operador de Seleção

A população inicial para o GA é um conjunto de indivíduos, em que cada indivíduo representa uma solução do problema, essas soluções são representadas conforme a subseção 3.2.1, o tamanho da população é definido por um parâmetro P_{size} de indivíduos. Para gerar esses indivíduos é feita uma distribuição aleatória de tarefas nas máquinas do problema e (VALLADA; RUIZ, 2011) utiliza uma heurística de refinamento chamada *Multiple Insertion* (MI) que de acordo com (VALLADA; RUIZ, 2011) foi proposta por (KURZ; ASKIN, 2001) e consiste em ordenar as tarefas da solução inicial de acordo com a matriz de processamento e *setup* e depois colocar cada tarefa em cada posição de cada máquina e essa tarefa será inserida na melhor posição encontrada, ou seja, a posição que resulta o menor *makespan* para a solução.

O Operador de Seleção escolhido pelos autores foi feito por meio de um torneio, em que um percentual da população, denotado como *pressure*, participará do torneio e a solução com o menor *makespan* vence o torneio e é selecionada, esse critério de avaliação é detalhado na seção 3.1.

3.2.4.2 Operador de Cruzamento e Operador de Mutação

Após utilizar o Operador de Seleção para escolher dois indivíduos da população, o Operador de Cruzamento vai ser aplicado de acordo com uma dada probabilidade P_c. O objetivo do Operador de Cruzamento é gerar dois bons indivíduos, chamados de descendência ou filhos, a partir dos dois indivíduos que foram selecionados, chamados de pais.

(VALLADA; RUIZ, 2011) utilizou o Operador de Mutação *One Point Order Crossover* adaptado para máquinas paralelas. Esse operador realiza as seguintes operações para gerar os dois filhos, para cada máquina do problema é escolhido um ponto aleatório p, assim cada máquina tem seu próprio p selecionado aleatoriamente, após selecionado o ponto p do Pai 1

para a máquina 1, as tarefas que estão na primeira posição da máquina até a posição p da máquina serão copiados para a máquina 1 do primeiro Filho, as tarefas da posição $p + 1$ até a posição final da máquina será copiado para o Filho 2, e assim sucessivamente para todas as máquinas do Pai 1.

Com a operação realizada utilizando as tarefas do Pai 1, os dois filhos que estão sendo gerados terão uma boa parte de suas tarefas alocadas nas máquinas, porém para alocar o restante das tarefas que estão faltando em cada filho é utilizado o Pai 2. Assim as tarefas que estão no Pai 2 e que ainda não foram inseridas nos Filhos, vão ser inseridas nas mesmas máquinas em que essas tarefas estavam no Pai 2. Ao invés de inserir em qualquer posição da máquina as tarefas que estão faltando, (VALLADA; RUIZ, 2011) realiza um processo de busca local limitada em que cada tarefa que vem do Pai 2 e será inserida nos filhos, será colocada em cada posição da máquina e ficará inserida na posição que gera o menor tempo de conclusão para aquela máquina. Cada tarefa passará por esse processo, e desse modo, todas as tarefas vindas do Pai 2 serão inseridas nas melhores posições das máquinas nos Filhos.

Vale ressaltar que no trabalho proposto por (VALLADA; RUIZ, 2011) ao gerar a descendência, os dois filhos somente serão adicionados na população se eles possuírem o valor do *makespan* melhor que o pior indivíduo da população, caso contrario o filho gerado vai ser rejeitado e não será inserido na população.

A Figura 8 ilustra esse processo de cruzamento utilizado por (VALLADA; RUIZ, 2011) com 2 máquinas e 12 tarefas, os tempos de preparação e processamento foram tirados para melhor visualização. São representados duas soluções escolhidas através do Operador de Seleção, e a Figura 8(a) mostra os pontos p selecionados aleatoriamente para cada um das máquinas do Pai 1, após copiar as tarefas do Pai 1 para os filhos, eles ficarão conforme mostra a Figura 8(b), em que no Filho 1 na máquina 1 recebeu as tarefas da posição 0 até a posição p, que no exemplo p é igual a 3, da máquina 1 do Pai 1, no Filho 2, a primeira máquina recebe as tarefas que estão da posição $p + 1$, no exemplo posição 4, até o fim da lista de tarefas do Pai 1, o mesmo processo acontece para a máquina 2 do Pai 1 em que as tarefas da posição 0 da lista até a posição p é inserida na mesma máquina no Filho 1 e as tarefas de $p + 1$ até o fim da lista de tarefas da máquina são inseridas no Filho 2. A segunda etapa do processo é inserir as tarefas que estão faltando em cada Filho, mas retirá-las do Pai 2, essas tarefas que veem do Pai 2 são inseridas nas mesmas máquinas que estavam no Pai 2, porém serão inseridas na melhor posição na máquina, isto é, a posição que faz com que o tempo de conclusão da máquina seja o menor. Após esses passos a descendência é gerada, a Figura 8(c) ilustra os dois Filhos gerados após o processo de cruzamento. Porém, eles só serão inseridos na população se o valor do *makespan* deles for melhor que o pior indivíduo da população.

Depois de a descendência ser gerada, o Operador de Mutação poderá ser aplicado considerando uma determinada probabilidade P_m. A mutação aplica o movimento de *Shift*, explicado na subseção 3.2.3.2, e ao utilizar o Operador de Mutação é possível ter diversidade

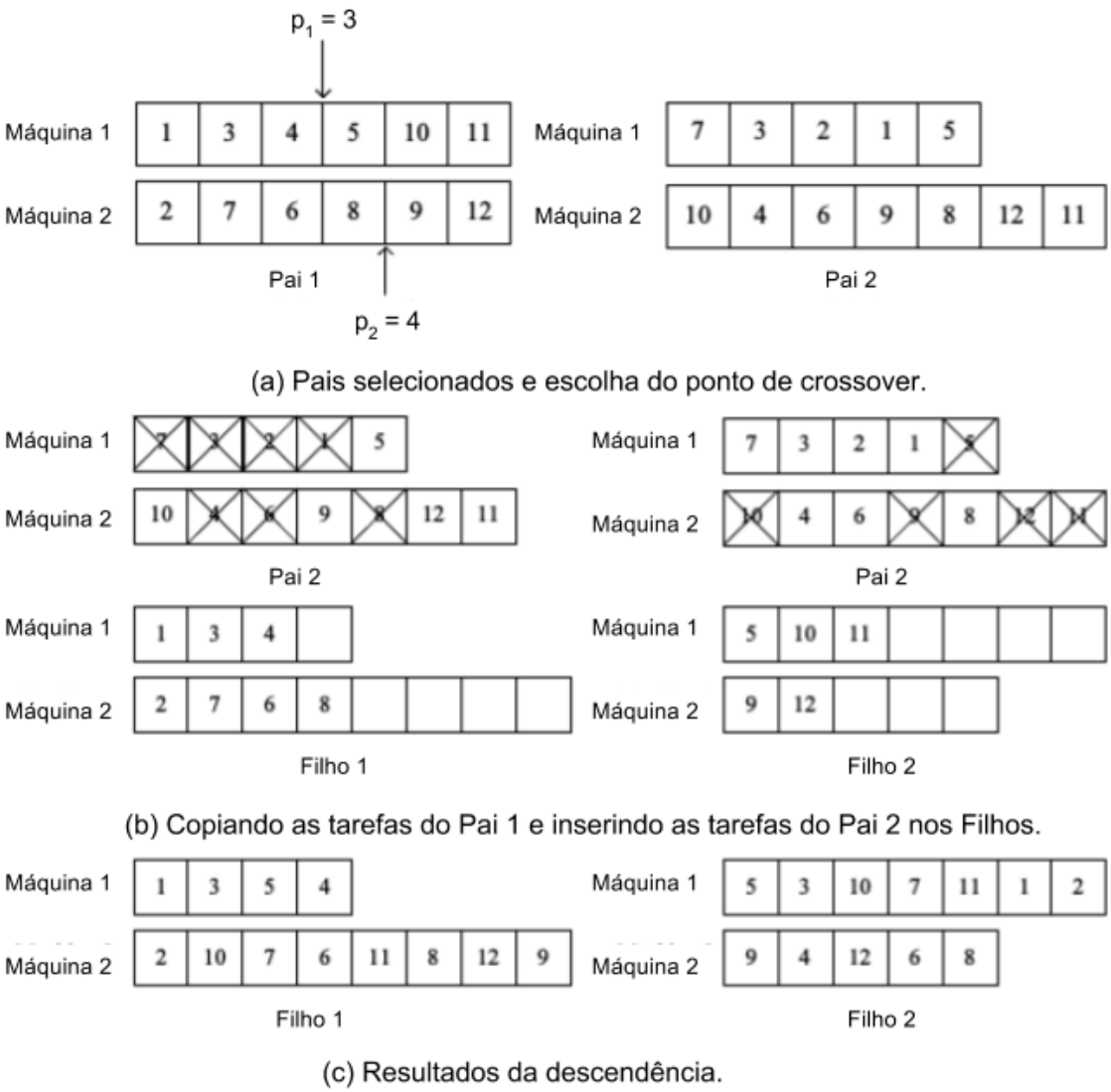


Figura 8: Exemplo da utilização do Operador de Cruzamento – (VALLADA; RUIZ, 2011)

na descendência gerada por meio da operação de cruzamento.

3.2.5 Critério de Parada

O critério de parada para tanto para o VNS quanto para o GA foi o mesmo usado por (VALLADA; RUIZ, 2011), a Equação 3.1 mostra como é feito o cálculo do critério de parada descrito pelos autores, sendo n o número de tarefas, m o número de máquinas e o parâmetro t varia entre $10, 30$ e 50.

$$stopExecution = n \times (m/2) \times t \; ms, \qquad t = \{10, 30, 50\} \tag{3.1}$$

3.3 Heurísticas Matemáticas

As heurísticas baseadas na formulação matemática do problema, utilizam o modelo matemático descrito na subseção 2.1.2, são abordadas a seguir. Ambas as heurísticas necessitam de um critério para o particionamento do conjunto de variáveis do problema, pois esse particionamento é base para essas heurísticas, e as estratégias de particionamento consideradas para esse trabalho, levando em conta as características do problema UPMSP, são descritas na subseção 3.3.4. Para resolver os subproblemas particionados é utilizado o resolvedor comercial *Gurobi* e soluções do problema são representadas no *Gurobi* conforme explicado na subseção 3.3.1.

3.3.1 Representação da Solução

A representação de uma solução no *Gurobi* é feita utilizando uma matriz de adjacência tridimensional, X, com o tamanho $m \times (n+1) \times (n+1)$, em que m é o número de máquinas e n o número total de tarefas do problema, e é somado um ao número de tarefas para representar as tarefas fictícias que vão preceder as primeiras tarefas do problema em cada máquina. Deste modo é possível representar qual tarefa vai ser executada em cada máquina e a ordem de precedência de execução de cada uma delas. A primeira dimensão da matriz representa as máquinas, e dentro de cada máquina existirá uma matriz $(n+1) \times (n+1)$, correspondente a segunda e terceira dimensão de X, em que as linhas representam as tarefas que foram executadas e as colunas são as tarefas que serão executadas imediatamente depois.

Pela definição do modelo matemático, seção 2.1, sempre é necessário que a tarefa que será executada atualmente tenha uma tarefa predecessora a ela, por esse motivo, a matriz começa com uma tarefa fictícia 0 e a função dessa tarefa fictícia é preceder a primeira tarefa do problema. Na matriz cada linha deve possuir apenas um valor 1 e os demais valores da linha assumir o valor 0 e cada coluna deve possuir também apenas um valor 1 e os demais valores da coluna serão 0, pois como definido na descrição do problema, seção 2.1, cada tarefa tem apenas uma predecessora, e é executada apenas uma tarefa por vez em cada máquina, então uma tarefa deve ser totalmente processada para que depois uma segunda tarefa possa ser executada na mesma máquina.

É ilustrado na Figura 9 a representação das variáveis do problema na matriz de adjacência X na modelagem para o resolvedor *Gurobi*. É representada a mesma solução que foi exemplificada na Figura 2 da seção anterior. As linhas, j_0 a j_6, são as tarefas que precedem as tarefas que serão executadas, representadas pelas colunas k_1 a k_6, por a coluna k_0 representar uma tarefa fictícia, toda a coluna assume valor 0. Assim a primeira máquina representada na figura é feita a seguinte leitura, a tarefa 6 é precedida por uma tarefa fictícia, 0, a tarefa 1 é executada imediatamente após a execução da tarefa 6 e a tarefa 3 será executada logo após a tarefa 1 ser executada. O mesmo se dá com as máquinas 2 e 3.

A avaliação das soluções é feita exatamente como foi descrito na seção 3.1.

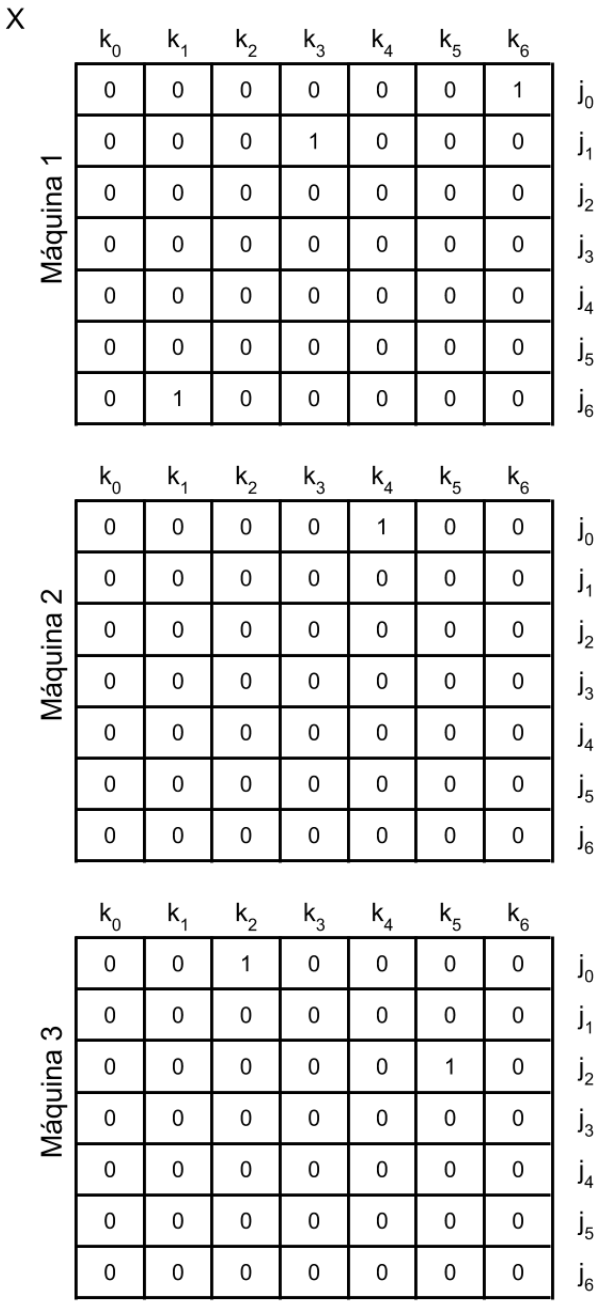


Figura 9: Representação de uma solução *s* do problema UPMSP no resolvedor *Gurobi*

3.3.2 *Relax-and-Fix*

De acordo com (OLIVEIRA, 2017) a heurística foi descrita por (WOLSEY, 1998) como um método que fragmenta o modelo de programação inteira mista em submodelos por meio do particionamento do conjunto das variáveis do modelo. Esses submodelos gerados são rapidamente resolvidos por causa da relaxação linear aplicada nas suas variáveis, assim é possível resolver o problema rapidamente, embora não seja garantido a solução ótima para o problema original. Essa heurística tem sido utilizada em diversos problemas de otimização combinatória, como problemas de dimensionamento de lotes ((BELVAUX; WOLSEY, 2000), (MERCÉ;

FONTAN, 2003), (AKARTUNALI; MILLER, 2009), entre outros) e também em problemas reais na indústria de bebidas (FERREIRA; MORABITO; RANGEL, 2010), fundição (ARAUJO; ARENALES; CLARK, 2008), papel e celulose (SANTOS; ALMADA-LOBO, 2012), no planejamento da produção em indústria química (CUNHA, 2013), entre outros (OLIVEIRA, 2017).

Na heurística *Relax-and-Fix* as variáveis do problema da matriz X são particionadas por um determinado critério de particionamento, e inicialmente todas as partições são relaxadas, ou seja, todas as variáveis do problema que inicialmente foram definidas como binárias se tornam contínuas. Após esse procedimento inicial, a heurística vai iterar por todas as partições e em cada iteração será realizado o seguinte procedimento: integralizar a partição, resolver o problema com uma partição binária e as demais contínuas e por fim fixa a partição que foi integralizada, nas iterações seguintes o mesmo processo se repete, porém, as partições fixadas não são mais alteradas. O Algoritmo 4 descreve esse método implementado nesse trabalho.

Algoritmo 4: Relax-and-Fix – R&F

```
1 Seja P um conjunto de partições das variáveis binárias do problema;
2 Relaxar todas as variáveis das partições do problema para serem contínuas;
3 para cada (partição i ∈ P) faça
4     Altera a partição i para ela voltar a ser binária;
5     Resolve o problema com a partição i binária e as demais contínuas;
6     Fixa as variáveis da partição i;
7 fim
```

Para facilitar o entendimento de como é feita essa iteração com as partições, a Figura 10 ilustra esse procedimento. Esse exemplo foi retirado de (CUNHA, 2013) para explicar o comportamento da heurística de modo simples e genérico. Primeiramente o problema original Figura 10(a) que contêm variáveis binárias é dividido em partições e essas partições são relaxadas, desse modo podem assumir valores contínuos, Figura 10(b), após essa divisão cada iteração da heurística vai integralizar uma partição e as demais vão continuar relaxadas, Figura 10(c), o problema é resolvido e a partição que foi integralizada é fixada, Figura 10(d), esse processo é repetido mas as partições fixadas não vão ser modificadas nas iterações futuras, até que todas as partições sejam integralizadas e fixadas, Figura 10(e), e ao fixar todas as partições a heurística termina. As partições destacadas em preto são as partições que foram fixadas durante as iterações da heurística.

A estratégia escolhida para ser usada no algoritmo implementado é o Particionamento por Tarefa, explicado na subseção 3.3.4.2, ela foi escolhida por gerar resultados melhores que as outras estratégias para essa heurística.

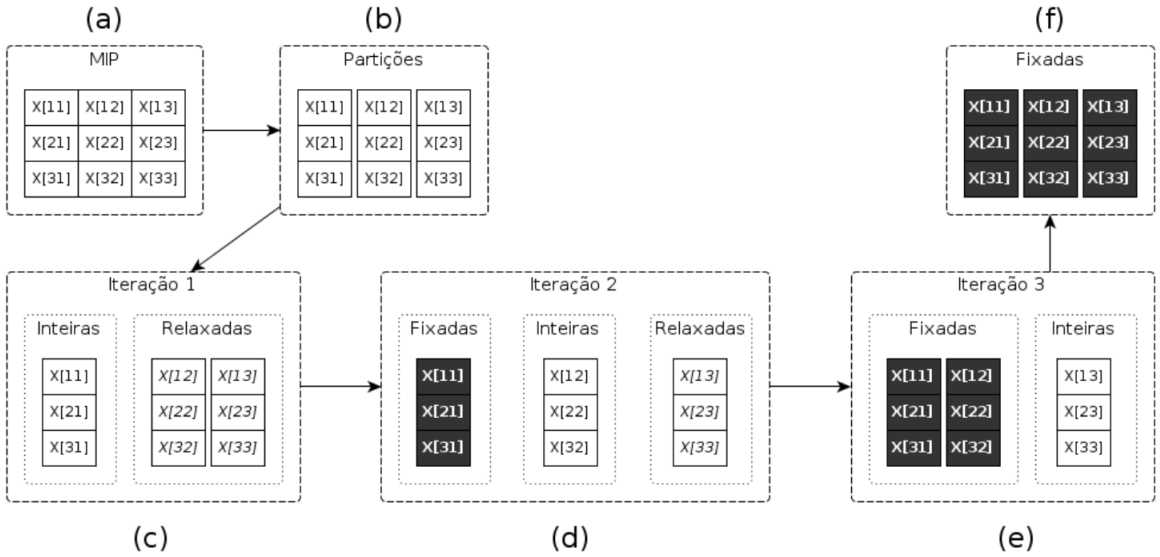


Figura 10: Representação genérica da heurística *Relax-and-Fix* – (CUNHA, 2013)

3.3.3 *Fix-and-Optimize*

Conforme (CUNHA, 2013) a heurística foi descrita por (POCHET; WOLSEY, 2006) inicialmente com o nome *exchange* e posteriormente chamada de *fix-and-optimize* em (SAHLING et al., 2009). A heurística tem por objetivo buscar novas soluções de melhor qualidade que a solução inicial fornecida. Essa heurística foi aplicada na solução de problemas de dimensionamento de lotes ((JAMES; ALMADA-LOBO, 2011), (LANG; SHEN, 2011), (SAHLING et al., 2009), (HELBER; SAHLING, 2010), (TOLEDO et al., 2015), entre outros).

Nessa heurística todas as variáveis binárias da matriz X são fixadas de acordo com uma solução inicial, depois é feito a divisão dessas variáveis em um conjunto de partições, com as partições definidas, a heurística realiza os seguintes passos: seleciona uma partição do conjunto de partições, faz essa partição que era fixada voltar a ser binária, resolve o problema com uma partição de variáveis binarias e as demais fixadas, fixa a partição novamente e repete esse processo na partição seguinte até que todas as partições passem por esse processo. O Algoritmo 5 descreve como esse processo foi implementado.

Algoritmo 5: Fix-and-Optimize – F&O

```
Dado uma solução viável do problema;
Seja P um conjunto de partições das variáveis binárias do problema original;
Fixar todas as variáveis das partições do problema;
para cada (partição i ∈ P) faça
    Altera a partição i para ela voltar a ser binária como no problema original;
    Resolve o problema com a partição i binária e as demais fixadas;
    se (solução for viável e melhor que atual) então
        Fixa a partição i;
    fim
    senão
        Descarta solução;
    fim
fim
```

Na Figura 11 esse procedimento é ilustrado para melhor entendimento do mesmo. Esse exemplo, como o da seção anterior, foi retirado de (CUNHA, 2013) com objetivo de exemplificar o comportamento da heurística de modo simples e genérico. O problema original com variáveis binárias é dividido em partições, depois todas as partições são fixadas, para representar essas partições fixadas elas foram destacadas em preto, e a cada iteração a heurística modifica as variáveis da partição para serem novamente binárias como no problema original e resolve o problema com essa partição binária e as demais fixadas, após a resolução do problema essa partição é novamente fixada e o processo se repete até que todas as partições tenham sido iteradas na heurística, conforme ilustrado na figura.

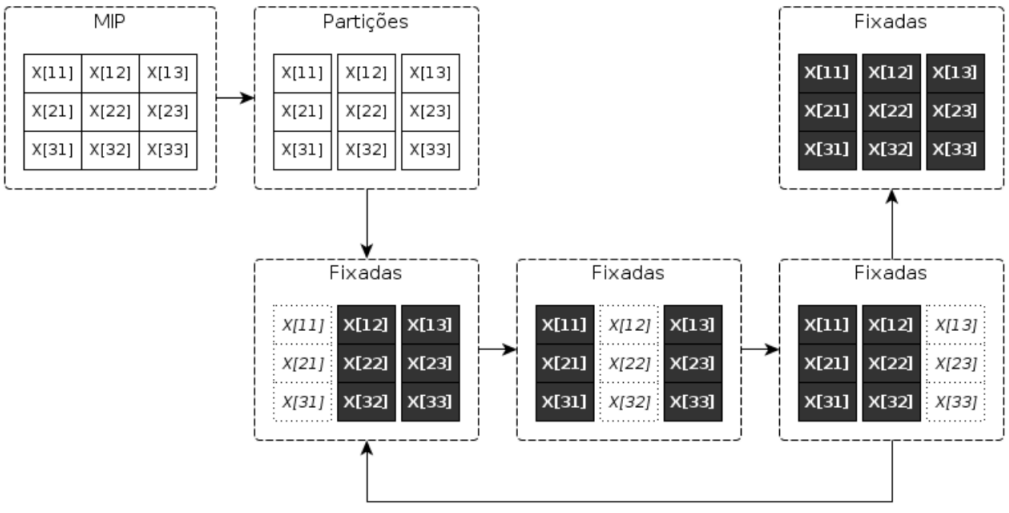


Figura 11: Representação genérica da heurística *Fix-and-Optimize* – (CUNHA, 2013)

A heurística *Fix-and-Optimize*, do mesmo modo que a *Relax-and-Fix*, necessita de

um critério de particionamento do conjunto de variáveis do problema. Dentre as estratégias de particionamento apresentadas na subseção 3.3.4, o Particionamento por Máquina e Tarefas foi escolhido para ser o critério de particionamento para a heurística implementada nesse trabalho, pois este tipo de particionamento mostrou ser melhor que os demais critérios de particionamentos apresentados na subseção 3.3.4.

3.3.4 Estratégias de Particionamento

Conforme descrito nas subseções anteriores, as heurísticas *Relax-and-Fix* e *Fix-and-Optimize* utiliza um critério para o particionamento do conjunto de variáveis para então atuarem sobre essas partições com objetivo de conseguir boas soluções, assim o modo como as partições são definidas interfere diretamente na qualidade da solução encontrada pelas heurísticas e por esse motivo, definir uma boa estratégia de particionamento de variáveis é fundamental e de grande importância para obter soluções de boa qualidade nas heurísticas.

Considerando as características do problema UPMSP, três estratégias de particionamento das variáveis foram examinadas, a subseção 3.3.4.1, subseção 3.3.4.2 e subseção 3.3.4.3 abordam cada uma delas. Para melhor entendimento dos particionamentos foram criadas ilustrações de cada tipo de particionamento utilizando a heurística F&O nos exemplos criados, e para que seja possível uma visualização mais clara, foi utilizada a mesma solução representada na subseção 3.3.1, com três máquinas e seis tarefas.

3.3.4.1 Particionamento por Máquina

Para o particionamento por máquina, o número de partições é definido pelo número de máquinas do problema, cada partição é uma máquina. Ao aplicar esse método de particionamento no F&O cada iteração liberará todas as variáveis de uma máquina (partição), com isso todas as suas variáveis que antes estavam fixadas, voltarão a ser binárias como no problema original e as demais variáveis das máquinas restantes continuarão fixadas conforme a solução inicial. A Figura 12 ilustra esse processo de particionamento, em que 3 partições são criadas, as variáveis que não estão na partição estão na cor cinza, essas variáveis estão fixadas, e as variáveis que fazem parte da partição, em branco, são binarias.

Com o particionamento por máquina é possível que as tarefas daquela partição fiquem livres para mudarem de ordem na sequência de execução de tarefas da máquina, o que afeta o tempo de conclusão daquela máquina, pois o tempo de *setup* das tarefas depende da ordem em que elas serão executadas dentro da máquina.

3.3.4.2 Particionamento por Tarefa

No particionamento por tarefa, a quantidade de partições é determinada pelo número de tarefas do problema, cada partição é uma tarefa. Usando a heurística F&O como exemplo para

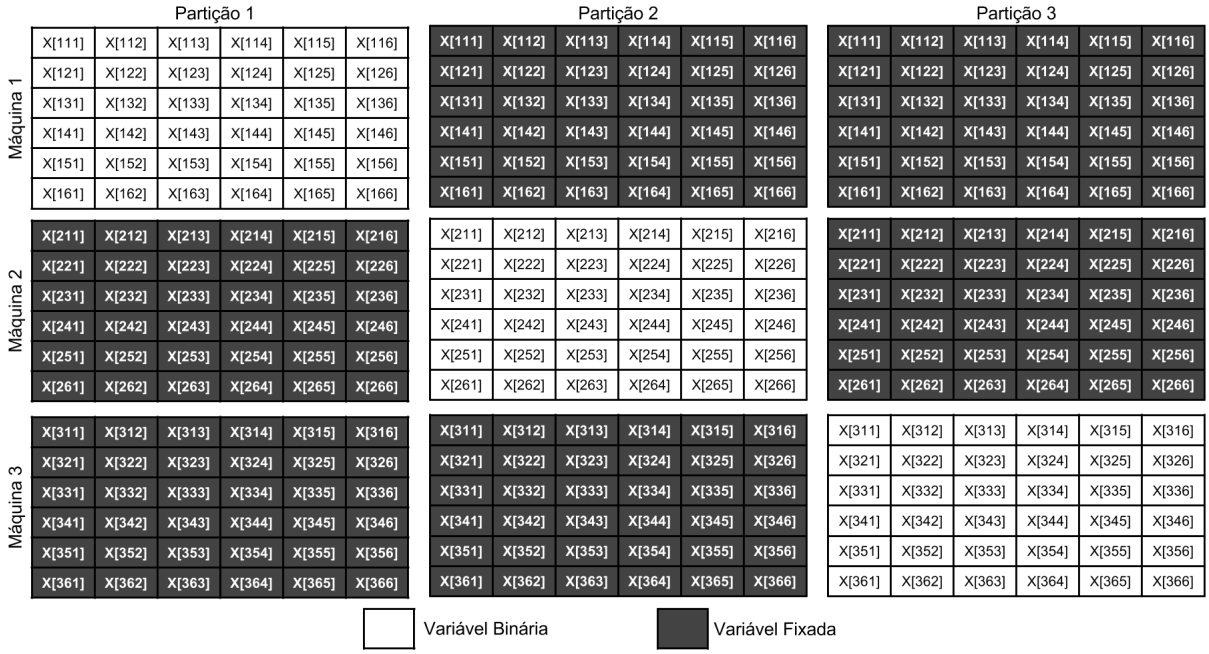


Figura 12: Representação do *Fix-and-Optimize* com partição por maquina.

esse particionamento, cada iteração da heurística fará com que partição (tarefa) seja liberada em todas as máquinas do problema, com essa estratégia de particionamento a tarefa poderá alternar entre as máquinas do problema, o que afeta o tempo de conclusão da máquina em que a tarefa estava antes da resolução do problema e a máquina em que a tarefa ficou alocada ao fim da resolução do problema. Todas as tarefas do problema em todas as máquinas são fixadas, apenas a tarefa da partição é liberada em todas as máquinas, conforme exemplificado na Figura 13, são criadas 6 partições e em cada partição um tarefa do problema é escolhida para ser liberada e voltar a ser binária e as demais, em cinza, permanecem fixadas de acordo com a solução inicial fornecida.

Nesse tipo de particionamento, o tempo de processamento e tempo de *setup* são levados em conta, pois a tarefa ao mudar de máquina, o tempo que a tarefa gasta para ser processada e o tempo de *setup* gasto pela máquina para ser configurada antes de executar essa tarefa, são diferentes em cada máquina.

3.3.4.3 Particionamento por Máquina e Tarefas

Nos particionamentos anteriores as tarefas podem mudar ou na mesma máquina ou entre outras máquinas, com o particionamento por máquina e tarefas as características dos dois tipos explicados na subseção 3.3.4.1 e subseção 3.3.4.2, são fundidos em uma única estratégia de particionamento. Assim nessa estratégia é possível obter ambos os benefícios tanto de as tarefas mudarem de posição na mesma máquina em que estão, tanto mudarem para outra máquina que produza um resultado de tempo de conclusão menor.

A Figura 14 exemplifica o uso desse particionamento na heurística F&O, considerando

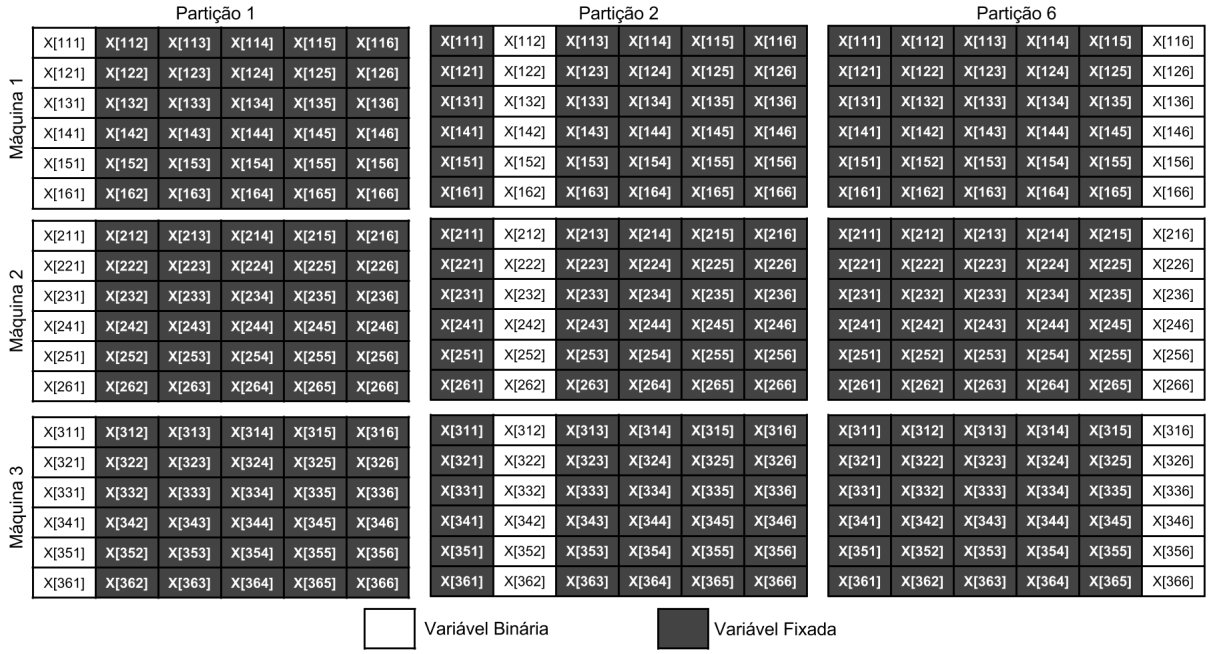


Figura 13: Representação do *Fix-and-Optimize* com partição por tarefa.

uma solução s com 3 máquinas e 6 tarefas e com as tarefas alocadas de acordo com a descrição feita na subseção 3.2.1, a partição 1 vai liberar todas as variáveis da máquina 1 e as tarefas que estão sendo processadas nessa máquina vão ser liberadas também nas demais máquinas do problema, assim as tarefas 1, 3 e 6 são liberadas na máquina 1 e também na máquina 2 e 3. Com esse tipo de particionamento a tarefa pode mudar de posição na sequência de processamento das tarefas da máquina em que ela estava inicialmente e também é possível o movimento entre máquinas.

3.3.5 Critério de Parada

Para as heurísticas matemáticas que utilizaram o resolvedor *Gurobi*, o critério de parada adotado foi de uma hora de tempo máximo de execução e depois de decorrido este tempo a melhor solução encontrada até então é retornada. (VALLADA; RUIZ, 2011) utilizou o mesmo critério de parada nos seus testes de instâncias pequenas utilizando o resolvedor CPLEX.

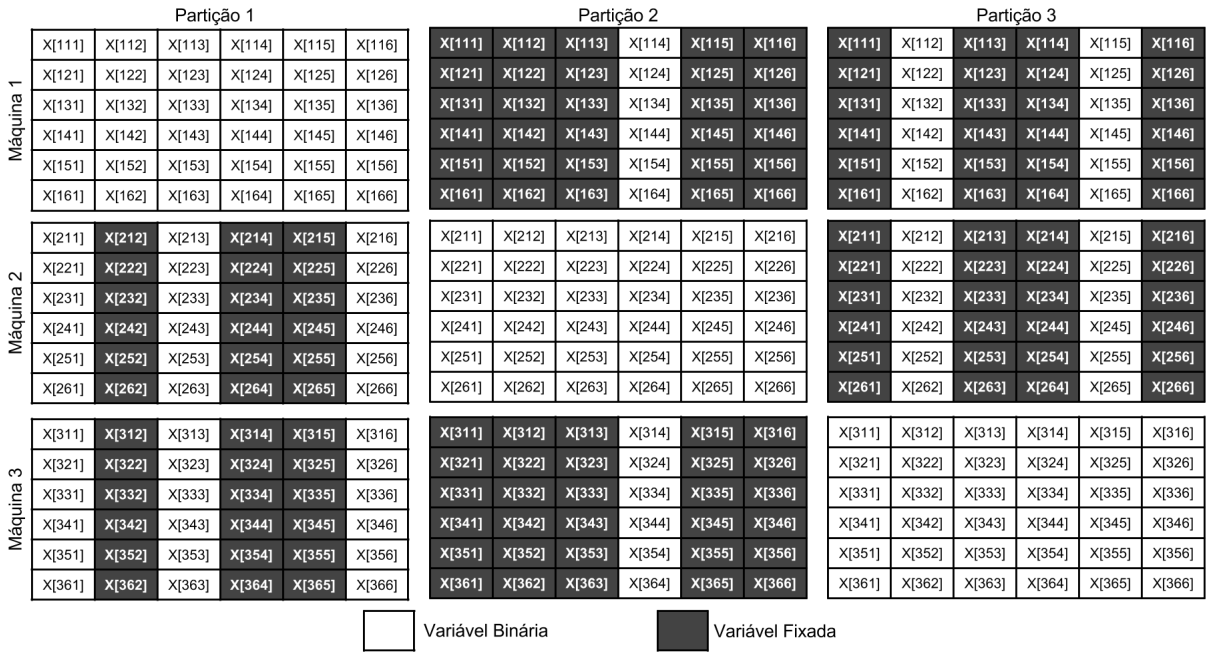


Figura 14: Representação do *Fix-and-Optimize* com partição por máquina e tarefas.

4 Resultados Obtidos

Neste Capítulo, a seção 4.1 descreve as instâncias utilizadas para testar os algoritmos implementados nesse trabalho e na seção 4.2 os resultados computacionais obtidos com a aplicação desses algoritmos.

Para realização de todos os testes foi utilizado um computador de arquitetura de 64 bits com processador *Intel® Core i7 CPU 870 @ 2.93GHz x 8*, 8 GB de memória RAM e sistema operacional *Linux, Ubuntu* 16.04. Foi utilizado o resolvedor comercial *Gurobi* na versão 7, com licença Acadêmica. Os algoritmos foram implementados na linguagem *C++*, os códigos foram compilados com o *GNU Compliler Collection*, versão 5.4.0.

4.1 Problemas-Teste

Para testar e avaliar os algoritmos implementados foram utilizadas instâncias criadas por (VALLADA; RUIZ, 2011), que possuem as seguintes características.

(VALLADA; RUIZ, 2011) criaram 1.640 instâncias que são divididas em dois conjuntos, o primeiro contendo 640 instâncias pequenas e no segundo conjunto, 1.000 instâncias grandes. Cada instância possui basicamente, m máquinas, n tarefas, uma matriz de tamanho $m \times n$ que define os tempos de processamento de cada tarefa j em cada máquina do problema e outra matriz de tamanho $m \times n \times n$ que determina o tempo de preparação quando uma tarefa j precede uma tarefa k numa máquina i. O primeiro conjunto de instâncias é formado pelas combinações de $n = \{6, 8, 10, 12\}$ e $m = \{2, 3, 4, 5\}$, já o segundo conjunto as combinações de tarefas e máquinas são $n = \{50, 100, 150, 200, 250\}$ e $m = \{10, 15, 20, 25, 30\}$.

As matrizes de tempos de processamento e de preparação, foram geradas por meio de uma distribuição uniforme. Com o tempo de processamento uniformemente distribuído entre 1 e 99, e para o tempo de preparação foi gerado 4 subconjuntos que foram uniformemente distribuídos entre 1–9, 1–49, 1–99 e 1–124.

Todas as instâncias podem ser encontradas em (SOA-ITI, 2013) com os melhores resultados de *makespan* obtidos para todas as instâncias.

4.2 Resultado Computacionais

Para testar os algoritmos foram usados dois conjuntos de instâncias, o primeiro conjunto com 16 instâncias pequenas, e o segundo com 25 instâncias grandes. Essas instâncias foram retiradas de cada combinação $n \times m$ na faixa restrita 1–9, para as instâncias pequenas o número de tarefas variou entre $n = \{6, 8, 10, 12\}$ e número de máquinas $m = \{2, 3, 4, 5\}$, já as instâncias

grandes o número de tarefas variou entre $n = \{50, 100, 150, 200, 250\}$ e número de máquinas $m = \{10, 15, 20, 25, 30\}$. Todas as instâncias usadas estão disponíveis (SOA-ITI, 2013).

A métrica utilizada para comparação dos algoritmos implementados é a mesma usada por (VALLADA; RUIZ, 2011), o Desvio de Porcentagem Relativa (RPD - *Relative Percentual Deviation*) e é dado pela Equação 4.1. Na equação, $Method_{sol}$ é o valor da função objetivo (*makespan*) da melhor solução encontrada por um dos algoritmos implementados e $Best_{sol}$ é o valor da função objetivo da melhor solução conhecida que se encontra em (SOA-ITI, 2013). O RPD determina a qualidade de uma solução em relação aos melhores resultados disponíveis em (SOA-ITI, 2013), assim quanto menor o valor do RPD maior é a qualidade da solução encontrada.

$$Relative\ Percentage\ Deviation\ (RPD) = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \times 100 \tag{4.1}$$

É possível concluir com a Equação 4.1 que valores negativos de RPD indicam que a solução encontrada pelo algoritmo superou a melhor solução encontrada em (SOA-ITI, 2013), porém valores positivos indicam o contrário.

Os resultados foram tabelados da seguinte forma, MIP referindo ao valores encontrados com a aplicação do método exato com a API do *Gurobi*, VNS a metaheurística *Variable Neighborhood Search*, GA o Algoritmo Genético, R&F representa os valores obtidos com a aplicação da heurística *Relax-and-Fix*, F&O VNS é a heurística *Fix-and-Optimize* com a solução inicial da heurística gerada a partir da aplicação do VNS nessa solução, e por fim, F&O GA a solução inicial da heurística é gerada após a aplicação do GA na solução.

Cada uma das instâncias do conjunto de instâncias testado, foi executada três vezes em cada algoritmo implementado e o melhor valor obtido por eles foi tabelado conforme descrito nas tabelas a seguir.

Nas tabelas 5, 6 e 7 tem-se na primeira coluna o nome da instância e os valores de RPD obtidos a partir dos resultados da aplicação dos algoritmos em cada instância do conjunto de instâncias pequenas testado. Nos algoritmos que foi adotado a Equação 3.1 como critério de parada, foram utilizados três valores para o parâmetro t dessa equação, esse parâmetro assumiu os valores 10, 30 e 50, e em cada uma das tabelas são apresentados os resultados dos algoritmos utilizando um desses valores. Vale ressaltar que esses três valores foram usados também por (VALLADA; RUIZ, 2011) nos seus algoritmos propostos.

Instância	MIP	VNS	GA	R&F	F&O VNS	F&O GA
6×2	0,00	0,00	0,00	68,92	0,00	0,00
6×3	0,00	0,00	0,00	1,92	0,00	0,00
6×4	0,00	0,00	0,00	9,09	0,00	0,00
6×5	0,00	0,00	0,00	0,00	0,00	0,00
8×2	0,00	0,00	0,00	43,13	0,00	0,00
8×3	0,00	0,00	0,00	202,86	0,00	0,00
8×4	0,00	0,00	0,00	33,02	0,00	0,00
8×5	0,00	0,00	0,00	280,00	0,00	0,00
10×2	0,00	0,00	0,00	27,43	0,00	0,00
10×3	0,00	0,00	0,00	182,28	0,00	0,00
10×4	0,00	0,00	0,00	80,60	0,00	0,00
10×5	0,00	0,00	0,00	123,08	0,00	0,00
12×2	0,00	0,00	0,00	51,98	0,00	0,00
12×3	0,00	0,00	0,00	43,93	0,00	0,00
12×4	0,00	0,00	0,90	81,08	0,00	0,90
12×5	0,00	0,00	0,00	136,25	0,00	0,00
Média	0,00	0,00	0,06	85,35	0,00	0,06

Tabela 5: Médias dos Desvios Percentuais Relativos (RPD) para os algoritmos implementados: com a variável t assumindo o valor 10 para o critério de parada utilizado nas metaheurísticas (instâncias pequenas)

De acordo com a Tabela 5, os algoritmos MIP, VNS, F&O VNS obtiveram RPD iguais a zero, isso significa que eles encontraram o valor ótimo para as instâncias do problema. Enquanto os algoritmos GA e F&O GA não conseguiram encontrar o ótimo apenas para a instância 12×4. E o algoritmo R&F obteve os piores resultados.

Conforme a Tabela 6, os algoritmos MIP, VNS, F&O VNS obtiveram RPD iguais a zero, ou seja, encontraram o valor ótimo para as instâncias testadas. Os algoritmos GA e F&O GA também encontram o valor ótimo para as instâncias, com exceção da instância 12×4. Enquanto o algoritmo R&F obteve os piores resultados para todas as instâncias.

Segundo a Tabela 7, apenas o algoritmo R&F obteve resultados ruins as instâncias, e os demais algoritmos, MIP, VNS, F&O VNS, GA e F&O GA, obtiveram os valores ótimos para todas as instâncias testadas.

Nas tabelas 8, 9 e 10 os RPDs para as instâncias grandes são apresentados. Da mesma forma que as instâncias pequenas, nos testes realizados com as instâncias grandes, cada algoritmo foi executado utilizando os três valores para o parâmetro t. Os campos das tabelas que não possuem valor numérico (–), são as instâncias que utilizaram o resolvedor *Gurobi* e embora o computador utilizado nos testes possuir mais de 20 GB de memória (memória RAM + swap), não foi possível retornar uma solução, pois toda a memória disponível para o resolvedor não foi suficiente.

Segundo a Tabela 8, o algoritmo MIP obteve o melhor resultado conhecido para as

Instância	MIP	VNS	GA	R&F	F&O VNS	F&O GA
6×2	0,00	0,00	0,00	68,92	0,00	0,00
6×3	0,00	0,00	0,00	1,92	0,00	0,00
6×4	0,00	0,00	0,00	9,09	0,00	0,00
6×5	0,00	0,00	0,00	0,00	0,00	0,00
8×2	0,00	0,00	0,00	43,13	0,00	0,00
8×3	0,00	0,00	0,00	202,86	0,00	0,00
8×4	0,00	0,00	0,00	33,02	0,00	0,00
8×5	0,00	0,00	0,00	280,00	0,00	0,00
10×2	0,00	0,00	0,00	27,43	0,00	0,00
10×3	0,00	0,00	0,00	182,28	0,00	0,00
10×4	0,00	0,00	0,00	80,60	0,00	0,00
10×5	0,00	0,00	0,00	123,08	0,00	0,00
12×2	0,00	0,00	0,00	51,98	0,00	0,00
12×3	0,00	0,00	0,00	43,93	0,00	0,00
12×4	0,00	0,00	2,53	81,08	0,00	2,53
12×5	0,00	0,00	0,00	136,25	0,00	0,00
Média	0,00	0,00	0,16	85,35	0,00	0,16

Tabela 6: Médias dos Desvios Percentuais Relativos (RPD) para os algoritmos implementados: com a variável *t* assumindo o valor 30 para o critério de parada utilizado nas metaheurísticas (instâncias pequenas)

Instância	MIP	VNS	GA	R&F	F&O VNS	F&O GA
6×2	0,00	0,00	0,00	68,92	0,00	0,00
6×3	0,00	0,00	0,00	1,92	0,00	0,00
6×4	0,00	0,00	0,00	9,09	0,00	0,00
6×5	0,00	0,00	0,00	0,00	0,00	0,00
8×2	0,00	0,00	0,00	43,13	0,00	0,00
8×3	0,00	0,00	0,00	202,86	0,00	0,00
8×4	0,00	0,00	0,00	33,02	0,00	0,00
8×5	0,00	0,00	0,00	280,00	0,00	0,00
10×2	0,00	0,00	0,00	27,43	0,00	0,00
10×3	0,00	0,00	0,00	182,28	0,00	0,00
10×4	0,00	0,00	0,00	80,60	0,00	0,00
10×5	0,00	0,00	0,00	123,08	0,00	0,00
12×2	0,00	0,00	0,00	51,98	0,00	0,00
12×3	0,00	0,00	0,00	43,93	0,00	0,00
12×4	0,00	0,00	0,00	81,08	0,00	0,00
12×5	0,00	0,00	0,00	136,25	0,00	0,00
Média	0,00	0,00	0,00	85,35	0,00	0,00

Tabela 7: Médias dos Desvios Percentuais Relativos (RPD) para os algoritmos implementados: com a variável *t* assumindo o valor 50 para o critério de parada utilizado nas metaheurísticas (instâncias pequenas)

Instância	MIP	VNS	GA	R&F	F&O VNS	F&O GA
50×10	62,69	4,48	129,85	925,37	4,48	129,85
50×15	33,33	5,56	166,67	1141,67	5,56	166,67
50×20	29,03	9,68	103,23	1116,13	9,68	103,23
50×25	0,00	0,00	336,36	1940,91	0,00	336,36
50×30	0,00	18,18	681,82	1936,36	18,18	663,64
100×10	61,83	-6,11	151,91	501,81	-6,11	151,91
100×15	72,06	7,35	138,24	758,82	7,35	138,24
100×20	100,00	6,52	215,22	1773,91	6,52	213,04
100×25	–	6,45	251,61	–	6,45	251,61
100×30	–	27,59	224,14	–	27,59	224,14
150×10	–	4,35	167,39	–	4,35	165,76
150×15	–	3,09	205,15	–	3,09	152,58
150×20	–	20,00	240,00	–	20,00	226,67
150×25	–	8,33	206,25	–	8,33	206,25
150×30	–	5,56	327,78	–	5,56	327,78
200×10	–	5,77	145,38	–	5,77	145,38
200×15	–	10,69	153,44	–	10,69	153,44
200×20	–	10,26	180,77	–	10,26	180,77
200×25	–	14,75	101,64	–	14,75	101,64
200×30	–	6,12	232,65	–	6,12	232,65
250×10	–	11,15	138,85	–	11,15	138,85
250×15	–	6,67	160,00	–	6,67	160,00
250×20	–	17,00	257,00	–	17,00	257,00
250×25	–	11,84	209,21	–	11,84	209,21
250×30	–	22,81	222,81	–	22,81	222,81
Média	44,87	9,52	213,89	1261,87	9,52	210,38

Tabela 8: Médias dos Desvios Percentuais Relativos (RPD) para os algoritmos implementados: com a variável t assumindo o valor 10 para o critério de parada utilizado nas metaheurísticas (instâncias grandes)

instâncias 50×25 e 50×30 porém esse algoritmo juntamente com o algoritmo R&F não conseguiram resolver todas as instâncias testadas devido a limitação de memória do computador usado para resolver essas instâncias. Os algoritmos VNS e F&O VNS obtiveram os melhores resultados comparados aos demais algoritmos testados e na instância 50×25 obtiveram o melhor resultado conhecido para a instância, disponível em (SOA-ITI, 2013) e na instância 100×10, os algoritmos superaram o valor da melhor solução conhecida. Os algoritmos GA e F&O GA obtiveram os piores resultados entre os algoritmos testados.

Na Tabela 9, os algoritmos MIP e R&F encontraram solução apenas para parte das instâncias testadas devido a limitação de memória do computador em que os testes foram executados. Os algoritmos GA e F&O GA obtiveram resultados ruins para todas as instâncias. Nos algoritmos VNS e F&O VNS os resultados obtidos foram os melhores comparados aos demais algoritmos e em alguns casos, como nas instâncias 50×15, 50×25, 50×30 e 150×25,

Instância	MIP	VNS	GA	R&F	F&O VNS	F&O GA
50×10	62,69	4,48	129,85	925,37	4,48	129,85
50×15	33,33	0,00	166,67	1141,67	0,00	166,67
50×20	29,03	6,45	103,23	1116,13	6,45	103,23
50×25	0,00	0,00	336,36	1940,91	0,00	318,18
50×30	0,00	0,00	654,55	1936,36	0,00	581,82
100×10	61,83	-6,11	119,08	501,81	-6,11	119,08
100×15	72,06	-1,47	138,24	758,82	-1,47	138,24
100×20	100,00	4,35	213,04	1773,91	4,35	213,04
100×25	–	9,68	251,61	–	9,68	251,61
100×30	–	17,24	224,14	–	17,24	224,14
150×10	–	2,72	169,02	–	2,72	169,02
150×15	–	2,06	215,46	–	2,06	215,46
150×20	–	13,33	242,67	–	13,33	242,67
150×25	–	0,00	206,25	–	0,00	206,25
150×30	–	5,56	330,56	–	5,56	330,56
200×10	–	2,69	145,38	–	2,31	145,38
200×15	–	6,87	154,20	–	6,87	152,67
200×20	–	3,85	179,49	–	3,85	179,49
200×25	–	9,84	101,64	–	9,84	101,64
200×30	–	6,12	232,65	–	6,12	232,65
250×10	–	6,83	137,77	–	6,83	137,77
250×15	–	4,67	152,00	–	4,67	152,00
250×20	–	12,00	257,00	–	12,00	257,00
250×25	–	6,58	209,21	–	6,58	209,21
250×30	–	15,79	221,05	–	15,79	221,05
Média	44,87	5,34	211,64	1261,87	5,33	207,95

Tabela 9: Médias dos Desvios Percentuais Relativos (RPD) para os algoritmos implementados: com a variável t assumindo o valor 30 para o critério de parada utilizado nas metaheurísticas (instâncias grandes)

os valores obtidos foram iguais aos melhores conhecidos disponíveis em (SOA-ITI, 2013) e nas instâncias 100×10 e 100×15, os algoritmos superaram o valor da melhor solução conhecida.

Na Tabela 10, é possível notar que os algoritmos VNS e F&O VNS obtiveram os melhores resultados de RPD, e nas instâncias 100×10, 100×15, 150×10 e 200×10, os algoritmos superaram a melhor solução conhecida, disponível em (SOA-ITI, 2013). O algoritmo F&O VNS também superou a solução conhecida para a instância 250×10. Os algoritmos MIP e R&F não conseguiram solucionar todas as instâncias testadas por causa da limitação de memória do computador usado nos testes. O algoritmo GA e F&O GA não obtiveram bons resultados, e o algoritmo R&F obteve os piores resultados.

As tabelas 11 e 12 comparam os valores de RPD obtidos com a aplicação da mataheurística VNS com os melhores valores de RPD apresentados por (VALLADA; RUIZ, 2011).

De acordo com a Tabela 11, o algoritmo VNS obteve valores de RPD melhores que os

Instância	MIP	VNS	GA	R&F	F&O VNS	F&O GA
50×10	62,69	4,48	71,05	925,37	4,48	60,53
50×15	33,33	0,00	166,67	1141,67	0,00	166,67
50×20	29,03	6,45	103,23	1116,13	6,45	103,23
50×25	0,00	0,00	264,86	1940,91	0,00	259,46
50×30	0,00	0,00	259,26	1936,36	0,00	248,15
100×10	61,83	-6,11	151,91	501,81	-6,11	142,75
100×15	72,06	-1,47	138,24	758,82	-1,47	138,24
100×20	100,00	0,00	215,22	1773,91	0,00	215,22
100×25	–	3,23	210,96	–	3,23	210,96
100×30	–	17,24	224,14	–	17,24	224,14
150×10	–	-2,72	111,45	–	-2,72	106,63
150×15	–	2,06	162,86	–	2,06	162,86
150×20	–	5,67	200,00	–	5,67	200,00
150×25	–	0,00	176,79	–	0,00	175,89
150×30	–	2,78	242,35	–	2,78	242,35
200×10	–	-1,20	139,57	–	-1,20	139,57
200×15	–	6,87	135,18	–	6,87	118,18
200×20	–	6,41	178,98	–	6,41	178,98
200×25	–	7,80	98,36	–	7,80	98,36
200×30	–	6,12	174,59	–	6,12	174,59
250×10	–	0,20	130,40	–	-0,20	130,40
250×15	–	0,32	158,67	–	0,00	158,67
250×20	–	12,00	134,05	–	12,00	134,05
250×25	–	6,58	169,71	–	6,58	169,71
250×30	–	15,79	197,83	–	15,79	197,83
Média	44,87	3,70	168,65	1261,87	3,67	166,30

Tabela 10: Médias dos Desvios Percentuais Relativos (RPD) para os algoritmos implementados: com a variável t assumindo o valor 50 para o critério de parada utilizado nas metaheurísticas (instâncias grandes)

apresentados apresentados por (VALLADA; RUIZ, 2011).

Conforme a Tabela 12, o algoritmo VNS obteve valores de RPD melhores que a maioria dos resultados encontrados por (VALLADA; RUIZ, 2011) e nas instâncias 100×10, 100×15, 150×10 e 200×10, o resultado encontrado pelo VNS superou a melhor solução conhecida para essas instâncias, disponível em (SOA-ITI, 2013).

Instância	VNS	(VALLADA; RUIZ, 2011)
6×2	0,00	0,00
6×3	0,00	0,08
6×4	0,00	0,37
6×5	0,00	0,12
8×2	0,00	0,00
8×3	0,00	0,31
8×4	0,00	0,41
8×5	0,00	0,11
10×2	0,00	0,07
10×3	0,00	0,18
10×4	0,00	0,30
10×5	0,00	1,03
12×2	0,00	0,10
12×3	0,00	0,12
12×4	0,00	0,89
12×5	0,00	1,49
Média	0,00	0,35

Tabela 11: Comparação entre o VNS e os melhores RPD encontrados por (VALLADA; RUIZ, 2011) (instâncias pequenas)

Instância	VNS	(VALLADA; RUIZ, 2011)
50×10	4,48	6,49
50×15	0,00	9,20
50×20	6,45	9,57
50×25	0,00	8,10
50×30	0,00	9,40
100×10	-6,11	5,54
100×15	-1,47	7,32
100×20	0,00	8,59
100×25	3,23	8,07
100×30	17,24	7,90
150×10	-2,72	5,28
150×15	2,06	6,80
150×20	5,67	7,40
150×25	0,00	7,05
150×30	2,78	7,17
200×10	-1,20	4,24
200×15	6,87	6,21
200×20	6,41	6,71
200×25	7,80	6,82
200×30	6,12	7,09
250×10	0,20	4,38
250×15	0,32	5,12
250×20	12,00	6,92
250×25	6,58	6,02
250×30	15,79	5,91
Média	3,70	6,93

Tabela 12: Comparação entre o VNS e os melhores RPD encontrados por (VALLADA; RUIZ, 2011) (instâncias grandes)

5 Considerações Finais

Na seção seção 5.1 é apresentada as conclusões que foram geradas por meio do desenvolvimento dessa monografia e a seção 5.2 aponta os possíveis trabalhos que podem ser realizados no futuro.

5.1 Conclusões

Este trabalho avaliou o modelo matemático, heurísticas matemáticas e metaheurísticas aplicadas ao problema UPMSP. No modelo matemático e nas heurísticas baseadas nesse modelo foi utilizado o resolvedor *Gurobi* e cinco estruturas de vizinhança para o VNS. E o algoritmo genético implementado adotou os parâmetros apresentados por (VALLADA; RUIZ, 2011).

A metaheurística VNS e a heurística F&O usando o VNS na solução inicial obtiveram os melhores resultados tanto para instâncias pequenas como grandes. É importante destacar que o uso de uma metaheurística simples como a VNS consegue ser muito eficiente e obtém bons resultados para o problema UPMSP com baixo custo computacional. Além disso, o uso de métodos baseados em formulação matemática demonstrou-se bastante eficiente quando aplicada à um problema *NP-Difícil*, como o UPMSP.

5.2 Trabalhos Futuros

Com objetivo de melhorar a performance dos algoritmos implementados aplicados ao problema UPMSP e obter melhores resultados para instâncias grandes, os seguintes trabalhos futuros podem ser realizados:

- Avaliar diferentes métodos para gerar uma solução inicial de melhor qualidade para o VNS;
- Avaliar a eficiência de cada uma das estruturas de vizinhança utilizadas nesse trabalho e também aplicar e avaliar outras estratégias de estruturas de vizinhanças, pois essas afetam diretamente na exploração do espaço de busca, e desse modo, são uma etapa crucial para encontrar soluções melhores e sair de ótimos locais na metaheurística VNS;
- Testar outros valores para os parâmetros do GA, diferentes dos propostos por (VALLADA; RUIZ, 2011), com objetivo de obter melhores resultados;
- Utilizar formulação de programação matemática diferente da proposta por (VALLADA; RUIZ, 2011), para o modelo MIP usado no resolvedor *Gurobi*, e avaliar como a diferença

na formulação matemática do problema UPMSP afeta na solução feita pelo *Gurobi* nas heurísticas *Fix-and-Optimize* e *Relax-and-Fix*;

- Aplicar e avaliar outros critérios de particionamento para as heurísticas *Fix-and-Optimize* e *Relax-and-Fix*;
- Pesquisar e aplicar no problema UPMSP outras heurísticas baseadas na formulação matemática do problema;
- Utilizar heurísticas como a *Iterated Local Search* (ILS), *Greed Randomized Adaptive Search Procedure* (GRASP) e *Simulated Annealing* (SA) na solução inicial antes de aplicar a heurística matemática *Fix-and-Optimize* e comparar com os resultados dessa heurística usando o VNS.

Referências

AKARTUNALI, K.; MILLER, A. J. A heuristic approach for big bucket multi-level production planning problems. *European Journal of Operational Research*, v. 193, n. 2, p. 396 – 411, 2009. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii-/S0377221707011307>. Citado na página 27.

ARAUJO, S. A. de; ARENALES, M. N.; CLARK, A. R. Lot sizing and furnace scheduling in small foundries. *Computers Operations Research*, v. 35, n. 3, p. 916 – 932, 2008. ISSN 0305-0548. Part Special Issue: New Trends in Locational Analysis. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054806001390>. Citado na página 27.

BELVAUX, G.; WOLSEY, L. A. bc — prod: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, v. 46, n. 5, p. 724–738, 2000. Disponível em: <https://doi.org/10.1287/mnsc.46.5.724.12048>. Citado na página 26.

CUNHA, A. L. da. Métodos heurísticos para um problema de planejamento da produção em uma indústria química. 2013. Citado 4 vezes nas páginas xv, 27, 28 e 29.

FERREIRA, D.; MORABITO, R.; RANGEL, S. Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants. *Computers Operations Research*, v. 37, n. 4, p. 684 – 691, 2010. ISSN 0305-0548. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054809001695>. Citado na página 27.

FLESZAR, K.; CHARALAMBOUS, C.; HINDI, K. A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 2011. Citado na página 11.

GENDREAU, M.; LAPORTE, G.; GUIMARÃES, E. M. A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 133, p. 183–189, 2001. Citado na página 2.

HADDAD, M. N. Algoritmos Heurísticos Híbridos para o Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de PReparação Dependentes da Sequência. p. 88, 2012. Disponível em: <http://www.decom.ufop.br/pos/site_media/uploads_ppgcc-/publicacao/haddad20>. Citado 2 vezes nas páginas 11 e 12.

HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search: Principles and applications. *European journal of operational research*, Elsevier, v. 130, n. 3, p. 449–467, 2001. Citado na página 15.

HANSEN, P.; MLADENOVIC, N. *A tutorial on variable neighborhood search*. [S.l.]: Groupe d'études et de recherche en analyse des décisions, HEC Montréal, 2003. Citado na página 15.

HELBER, S.; SAHLING, F. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, Elsevier, v. 123, n. 2, p. 247–256, 2010. Citado na página 28.

JAMES, R. J.; ALMADA-LOBO, B. Single and parallel machine capacitated lotsizing and scheduling: New iterative mip-based neighborhood search heuristics. *Computers & Operations Research*, Elsevier, v. 38, n. 12, p. 1816–1825, 2011. Citado na página 28.

KIM, D. et al. Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, Elsevier Limited, v. 18, n. 3-4, p. 223–231, 6 2002. ISSN 0736-5845. Citado na página 2.

KURZ, M. E.; ASKIN, R. G. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, v. 39, n. 16, p. 3747–3769, 2001. Disponível em: <http://dx.doi.org/10.1080/00207540110064938>. Citado na página 22.

LANG, J. C.; SHEN, Z.-J. M. Fix-and-optimize heuristics for capacitated lot-sizing with sequence-dependent setups and substitutions. *European Journal of Operational Research*, Elsevier, v. 214, n. 3, p. 595–605, 2011. Citado na página 28.

MERCÉ, C.; FONTAN, G. Mip-based heuristics for capacitated lotsizing problems. *International Journal of Production Economics*, v. 85, n. 1, p. 97 – 111, 2003. ISSN 0925-5273. Planning and Control of Productive Systems. Disponível em: <http://www.sciencedirect.com-/science/article/pii/S0925527303000902>. Citado na página 27.

MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 3. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 1996. 387 p. ISBN 978-3-540-60676-5. Citado 2 vezes nas páginas 20 e 21.

OLIVEIRA, J. D. de. ESTRATÉGIAS RELAX-AND-FIX APLICADA AO PROBLEMA DE ROTEAMENTO EM ARCOS CAPACITADO E PERIÓDICO. p. 98, 2017. Citado 2 vezes nas páginas 26 e 27.

PINEDO, M. L. *Scheduling Theory, Algorithms, and Systems*. [S.l.: s.n.], 2008. 665 p. ISBN 9780387789347. Citado 2 vezes nas páginas 1 e 5.

POCHET, Y.; WOLSEY, L. A. *Production planning by mixed integer programming*. [S.l.]: Springer Science & Business Media, 2006. Citado na página 28.

RABADI, G.; MORAGA, R.; AL-SALEM, A. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, v. 17, p. 85–97, 2006. Citado na página 11.

RAVETTI, M. G. Algoritmos para o Problema de Seqüenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Seqüência. 2007. Citado 3 vezes nas páginas 2, 10 e 16.

SAHLING, F. et al. Solving a multi-level capacitated lot sizing problem with multi-period setup carry-over via a fix-and-optimize heuristic. *Computers & Operations Research*, Elsevier, v. 36, n. 9, p. 2546–2553, 2009. Citado na página 28.

SANTOS, M. O.; ALMADA-LOBO, B. Integrated pulp and paper mill planning and scheduling. *Computers Industrial Engineering*, v. 63, n. 1, p. 1 – 12, 2012. ISSN 0360-8352. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0360835212000204>. Citado na página 27.

SILVA, C. L. T. d. F. e. Metaheurísticas de Busca Local para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependente da Sequência. 2014. Citado 11 vezes nas páginas xv, xvii, 6, 7, 8, 10, 11, 16, 18, 19 e 20.

SOA-ITI. Grupo de investigación SOA-ITI: Sistemas de Optimización Aplicada - Instituto Tecnológico de Informática. 2013. Disponível em: <http://soa.iti.es/problem-instances>. Acesso em: 12 de maio de 2017. Citado 9 vezes nas páginas xi, xiii, 1, 12, 35, 36, 39, 40 e 41.

STEFANELLO, F. et al. MIP–Based Neighborhood Search for the Unrelated Parallel Machine Scheduling Problem With Sequence and Machine-Dependent Setup Times. *International Journal of Advanced Manufacturing Technology*, 2015. Citado na página 12.

TANG, L.; WANG, X. Simultaneously scheduling multiple turns for steel color-coating production. *European Journal of Operational Research*, v. 198, n. 3, p. 715–725, 2009. Disponível em: <https://doi.org/10.1016/j.ejor.2008.09.025>. Citado na página 2.

TOLEDO, C. F. M. et al. A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. *Journal of Heuristics*, Springer, v. 21, n. 5, p. 687–717, 2015. Citado na página 28.

VALLADA, E.; RUIZ, R. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, Elsevier B.V., v. 211, n. 3, p. 612–622, 2011. ISSN 03772217. Disponível em: <http://dx.doi-.org/10.1016/j.ejor.2011.01.011>. Citado 25 vezes nas páginas xi, xiii, xv, xvii, 1, 2, 6, 8, 10, 11, 12, 13, 16, 21, 22, 23, 24, 32, 35, 36, 40, 41, 42, 43 e 45.

WOLSEY, L. A. *Integer Programming*. [S.l.: s.n.], 1998. 288 p. ISBN 978-0-471-28366-9. Citado na página 26.