Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM

Otimização de Linhas de Montagem Com Máquinas Flexíveis Paralelas Idênticas

Bolsista: Túlio Neme de Azevedo
Orientador: Marco Antonio Moreira de Carvalho – DECOM/UFOP

Nota: Relatório Final referente ao período de dd/MM/AAAA a dd/MM/AAAA, apresentado à Universidade Federal de Ouro Preto, como parte das exigências do programa ...

Local: Ouro Preto - Minas Gerais - Brasil
Data: 20 de novembro de 2017

Título do Resumo

Assinatura do orientador(a): _____
Marco Antonio Moreira de Carvalho

Assinatura do bolsista: _____
Túlio Neme de Azevedo

Sumário

Capítulo 1

Introdução

Um *Sistema de Manufatura Flexível* se caracteriza pela flexibilidade de adaptação caso haja alterações no planejamento da produção. O avanço tecnológico atual possibilita a resolução de problemas operacionais utilizando métodos computacionais em sistemas que utilizam máquinas flexíveis, entre outros. O resultado esperado pelo uso desses métodos é uma linha de produção mais eficiente em termos de produtividade.

Uma *máquina flexível* é caracterizada por dinamizar uma linha de produção, devido à sua capacidade de produzir diversos produtos não relacionados entre si. Para produzir cada produto é necessário que um conjunto específico de *ferramentas* esteja carregada na máquina, que por sua vez possui uma capacidade limitada de ferramentas carregadas simultaneamente. A capacidade da máquina é suficiente para comportar as ferramentas necessárias a qualquer produto isoladamente, porém não é suficiente para comportar todas as ferramentas disponíveis simultaneamente.

Para processar todo o conjunto de produtos designados à máquina flexível, trocas de ferramentas são necessárias, para que, entre o processamento de dois produtos diferentes subsequentes, a capacidade da máquina não seja excedida. Desta forma, ao realizar trocas de ferramentas, toda a produção é interrompida, gerando ociosidade na linha de produção e, por consequência, diminuindo sua eficiência.

No cenário referente à linhas de manufatura flexível, alguns problemas combinatórios são encontrados, tais como o Problema de Minimização de Troca de Ferramentas (*Minimization of Tool Switches Problem*, MTSP) e o Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas (*Identical Parallel Machines Problem with Tooling Constraints*, IPMTC). Para os problemas citados, considera-se que a produção de um produto como uma *tarefa* a ser realizada.

O MTSP tem como objetivo sequenciar os tarefas em um única máquina flexível de forma a minimizar a troca de ferramentas durante a produção, tornando-a mais eficiente. O IPMTC, por sua vez, pode ser considerado como uma extensão do MTSP, já que também tem a máquina flexível como ator principal, porém, este considera que p máquinas flexíveis idênticas estão disponíveis na linha de produção para que o processamento das tarefas seja realizado. Considera-se também que cada tarefa possui um tempo de processamento específico e que as trocas de ferramentas exigem uma quantidade uniforme de tempo. Desta forma, o objetivo relacionado ao IPMTC é minimizar o tempo total de produção. Para tal, considera-se que qualquer máquina tem poder de processamento igual e que cada uma delas é capaz de processar quaisquer produtos. Além disso, cada máquina possui suas próprias ferramentas, assim não há o compartilhamento de ferramentas entre as máquinas, e cada tarefa é processada unicamente em uma das máquina disponíveis.

De acordo com Fathi and Barnette (2002), o IPMTC se divide em três subproblemas: (i) alocar as tarefas às máquinas disponíveis; (ii) determinar o sequenciamento correto das tarefas em cada máquina; e (iii) determinar a alocação das ferramentas no compartimento de cada máquina. O primeiro subproblema é conhecido como Problema de Sequenciamento em Máquinas Paralelas (*Parallel Machine Scheduling Problem*, PM). O segundo e terceiro subproblemas, em conjunto, caracterizam o MTSP. Especificamente, o terceiro subproblema pode ser resolvido em tempo determinístico polinomial pela política *Keep Tool Neeeded Soonest* (KTNS) (Tang and Denardo, 1988). Quando trocas de ferramentas são necessárias, a política KTNS garante que as ferramentas que serão necessárias mais brevemente pelas tarefas ainda não processadas sejam mantidas na máquina. Definidos por Garey and Johnson (1979) e Crama et al. (1994), PM e MTSP, respectivamente, possuem complexidade $\mathcal{NP}$-Difícil. Quando o IPMTC possui uma única máquina flexível disponível, ele se reduz ao MTSP. Quando o IPMTC não necessita de troca de ferramentas, ele se reduz ao PM. Logo, por associação, o IPMTC também possui complexidade $\mathcal{NP}$-Difícil, Beezão et al. (2017).

Este trabalho tem como objetivo o estudo do IPMTC no caso geral, que aborda máquinas idênticas, processando tarefas de forma paralela, cujo objetivo é minimizar o tempo de conclusão de todo o processo. Além disso, considera-se que todas as ferramentas possuem tamanho e custo uniformes, ou seja, ocupem apenas uma posição no compartimento, que a posição ocupada não tenha relevância e que os tempos de troca de ferramentas sejam iguais em cada caso.

1.1 Definição do Problema

Formalmente, o IPMTC é definido da seguinte forma: dados um conjunto de máquinas flexíveis idênticas $P = [1,\ldots,p]$, com capacidade de comportar até C ferramentas simultaneamente, um conjunto de tarefas $T = [1,\ldots,n]$, um conjunto de ferramentas $F = [1,\ldots,m]$, o subconjunto de ferramentas F_i ($F_i \in F$) necessárias para realizar a tarefa i ($i \in T$) e tempo de processamento de cada tarefa, indicado por t_i ($i \in T$). Assim, é necessário selecionar quais as tarefas T direcionadas à cada máquina k ($k \in P$), de forma que o sequenciamento ϕ_k das tarefas em cada máquina k possua o menor número de trocas de ferramentas, resultando no menor tempo de produção. Para tal, o objetivo é determinar o menor *makespan* possível, dentre os sequenciamentos ϕ_k, representado por Δ. O *makespan* é o tempo de término de processamento mais longo dentre as máquinas disponíveis.

Na Tabela 1.1 é apresentado uma exemplo de instância para o IPMTC, indicando quais ferramentas são necessárias para executar cada tarefa. No cenário referido pela tabela, tem-se $p = 2$, $n = 5$, $m = 5$ e $C = 3$. Além disso, o tempo ($\bar{t}$) necessário para realizar a troca de duas ferramentas quaisquer é de 15 unidades de tempo.

Tabela 1.1: Exemplo de instância IPMTC.

Tempo de troca de ferramentas ($\bar{t}$) = 15		
Tarefas	Tempo de Processamento	Ferramentas
1	10	2, 3, 5
2	56	1, 3
3	15	1, 4, 5
4	48	1, 2
5	20	2, 3, 4

Na referida Tabela 1.1, a primeira coluna representa as tarefas a serem processadas pela máquina flexível, enumeradas de 1 a 5; a segunda coluna representa o tempo de processamento

total da tarefa; e a terceira coluna representa as ferramentas necessárias para que a tarefa referida seja processada. Computacionalmente, esta instância pode ser modelada por uma matriz binária Q, conforme apresentado pela Tabela 1.2. As linhas representam as ferramentas e as colunas representam as tarefas, ambas numeradas de 1 a 5. Os elementos q_{ij} da matriz Q são preenchidos da seguinte maneira: $q_{ij} = 1$ caso a ferramenta i ($i \in F$) seja necessária para processar a tarefa j ($j \in T$). Caso contrário, $q_{ij} = 0$.

Tabela 1.2: Matriz Q.

Ferramentas/Tarefas	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	1
3	1	1	0	0	1
4	0	0	1	0	1
5	1	0	1	0	0

Conforme dito anteriormente, após modelar o problema observa-se que o problema se decompõem em três etapas: alocação de tarefas, sequenciamento das tarefas em cada máquina e alocação de ferramentas em cada máquina. A Tabela 1.3 exemplifica a alocação de tarefas às duas máquina compostas pelo exemplo. A Tabela 1.3 apresenta as tarefas que foram alocadas para as máquinas 1 e 2. Para a máquina 1, foram alocado as tarefas 1, 3 e 4, ao passo que as tarefas 2 e 5 foram alocadas à máquina 2.

Tabela 1.3: Alocação das tarefas nas máquinas.

Máquinas	Tarefas
1	1, 3, 4
2	2, 5

Dada a alocação das tarefas às máquinas, é necessário determinar o sequenciamento ϕ_k ($k \in P$), das tarefas que minimize o número de troca de ferramentas, definido pela etapa seguinte. Este objetivo se refere ao MTSP. Desta forma, a Tabela 1.4 evidencia o sequenciamento de tarefas para cada máquina, sendo $\phi_1 = [1,4,3]$ resultando em 2 trocas de ferramentas e $\phi_2 = [2,5]$ resultando em 1 troca de ferramentas, representando a máquina 1 e 2 respectivamente.

Tabela 1.4: ϕ_1 (a) e ϕ_2 (b).

Ferramentas/ϕ_1	1	4	3
1	0	1	1
2	1	1	0
3	1	0	0
4	0	0	1
5	1	1	1

Ferramentas/ϕ_2	2	5
1	0	1
2	1	0
3	1	1
4	1	1
5	0	0

Considerando as três inserções das ferramentas iniciais em cada uma das máquinas, contabiliza-se 5 trocas para máquina 1 e 4 trocas na máquina 2.

O tempo gasto para o processamento de ϕ_1 são 148 unidades de tempo, divididos da seguinte forma:

- 45 unidades de tempo na inserção de três ferramentas iniciais (2, 3 e 5);

- 10 unidades de tempo no processamento da tarefa 1;
- 15 unidades de tempo na troca da ferramenta 3 pela 1, após o processamento da tarefa 1;
- 48 unidades de tempo no processamento da tarefa 4;
- 15 unidades de tempo na troca da ferramenta 2 pela 4, após o processamento da tarefa 4;
- 15 unidades de tempo no processamento da tarefa 3.

.

Já ϕ_2 necessita de 136 unidades de tempo para ser concluído e é dividido conforme descrição a seguir:

- 30 unidades de tempo na inserção de duas ferramentas iniciais (2e 3);
- 20 unidades de tempo no processamento da tarefa 5;
- 30 unidades de tempo na troca da ferramenta 2 pela 1 e da inserção da ferramenta 4, após o processamento da tarefa 5;
- 56 unidades de tempo no processamento da tarefa 2.

Considerando a representação por matrizes, uma solução ϕ_k induz uma matriz permutação R^{ϕ_k}=$\{r^{\phi_k}\}$, cujas colunas são as colunas de Q na ordem estabelecida por ϕ_k. Uma coluna adicional 0 com todos os elementos nulos é adicionada à esta matriz, representando o estado inicial da máquina, sem nenhuma ferramenta carregada. O valor dos elementos r^{ϕ_k}, onde k representa uma das máquinas flexíveis disponíveis, são definidos de acordo com a Equação 1.1.

$$r_{ji}^{\phi_k} = \begin{cases} 1, \text{ se a ferramenta } j \in F \text{ estiver na máquina } k \text{ durante o processamento da tarefa } i \in T \\ 0, \text{caso contrário.} \end{cases}$$

Na referida Tabela 1.5, apresenta-se as matrizes correspondentes às soluções ϕ_1 e ϕ_2 mencionadas anteriormente. As ferramentas carregadas a cada instante na máquina estão sublinhadas e as ferramentas que permanecem na máquina, mesmo sem serem utilizadas, estão indicadas em negrito. Por exemplo, após o processamento da tarefa 1 na máquina 1, a ferramenta 1 foi carregada, portanto, sublinhada, e a ferramenta 5 foi mantida na máquina mesmo sem ser utilizada pela tarefa 4, portanto, destacada em negrito.

Tabela 1.5: R^{ϕ_1} (a) e R^{ϕ_2} (b) ordenadas.

Ferramentas/ϕ_1	0	1	4	3
1	0	0	$\underline{1}$	1
2	0	$\underline{1}$	1	0
3	0	$\underline{1}$	0	0
4	0	0	0	$\underline{1}$
5	0	$\underline{1}$	**1**	1

Ferramentas/ϕ_2	0	2	5
1	0	0	$\underline{1}$
2	0	$\underline{1}$	0
3	0	$\underline{1}$	1
4	0	0	$\underline{1}$
5	0	0	0

Uma solução representada pela matriz R^{ϕ_k} é avaliada de acordo com a Equação 1.1, proposta por Crama et al. (1994). Esta função calcula, dada uma sequência de tarefas, o número de inversões de 0 para 1, que representam a inserção de ferramentas na máquina. Por exemplo,

vide os dados contidos na Tabela 1.4 (*a*), cujo número de trocas é 5. Esse cálculo faz referencia ao MTSP, uma vez que é a função objetivo do mesmo.

$$Z_{MTSP}(R^{\phi_k}) = \sum_{i \in T} \sum_{j \in F} r_{ji}^{\phi_k} (1 - r_{ji-1}^{\phi_k}) \tag{1.1}$$

O objetivo do IPMTC é minimizar o *makespan* (Δ), resultado da Equação 1.2. Essa função realiza a soma entre o tempo de processamento da tarefa (t_i) e a multiplicação entre a quantidade de trocas de ferramentas durante o processamento da tarefa corrente ($Z_{MTSP}(R^{\phi_k})$) e o tempo de troca entre duas ferramentas ($\bar{t}$).

$$\Delta = argmax\{ \sum_{i \in \phi_k} t_i + Z_{MTSP}(R^{\phi_k}) * \bar{t} \} \ \forall k \in P \tag{1.2}$$

Capítulo 2

Objetivos

O trabalho de pesquisa abordado neste relatório se propõe a estudar o Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas no contexto de sistemas de manufatura flexíveis, revisitando sua modelagem pelo Problema de Minimização de Troca de Ferramentas e utilizando o atual estado da arte como parte da solução deste problema.

Objetivos específicos:

1. Realizar pesquisa para geração de embasamento teórico e revisão bibliográfica sobre o tema tratado;
2. Realizar pesquisa sobre a modelagem do Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas como o Problema de Minimização de Trocas de Ferramentas;
3. Propor um método heurístico para solução do Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas; e
4. Avaliar o método implementado considerando problemas teste publicamente disponíveis, realizando uma análise crítica considerando outros métodos da literatura.

Capítulo 3

Revisão da Literatura

Os conceitos básicos deste trabalho foram fundamentados através das pesquisas anteriores referentes ao IPMTC. Dado que a literatura sobre o tema seja restrita, é possível descrever em maior profundidade cada um dos trabalhos encontrados nela. As seções a seguir, descrevem em ordem cronológica estes trabalhos.

3.1 O Trabalho Seminal de Stecke (1983)

O primeiro autor a abordar o IPMTC foi Stecke (1983), o qual avaliou os problemas em sistemas de manufatura flexível. Considerando que o problema de configuração geral de um SMF é intratável, foram definidos cinco problemas de planejamento de produção em um SMF. São eles:

1. Problema de Seleção de Tarefas;
2. Problema de Agrupamento de Máquinas;
3. Problema de Proporção de Produção;
4. Problema de Alocação de Recursos; e
5. Problema de Carregamento de Operações e Ferramentas.

Especificamente, os autores se concentraram no estudo do segundo e do quinto problema. O segundo problema se carateriza pelo agrupamento de máquinas idênticas disponíveis no SMF, de maneira que cada grupo em particular é capaz de realizar o mesmo conjunto de operações. O quinto problema é caracterizado pela alocação de operações e ferramentas às máquinas de um conjunto pré-determinado e considerando restrições de capacidade do SMF. A união destes dois problemas em particular indica uma formulação próxima da definição atual do IPMTC.

Para solução destes problemas foram propostas formulações de Programação Não Linear Inteira Mista (PNLIM), posteriormente linearizadas por diferentes métodos. Para o Problema de Agrupamento de Máquinas, foi realizada uma adaptação da formulação proposta anteriormente por Stecke and Solberg (1981). Esta adaptação deveu-se ao fato de o novo problema de agrupamento ser mais complexo que o anterior, sendo necessária a inclusão de restrições referentes aos requerimento de ferramentas pelas tarefas e quanto à capacidade das máquinas, além de ser necessário considerar os tempos de processamento das tarefas.

Para o Problema de Alocação de Operações e Ferramentas foram elencados seis possíveis objetivos em uma nova formulação, cada um aplicável a um contexto diferente, como por exemplo o equilíbrio do tempo de processamento de cada máquina ou atendimento prioritário a determinadas operações. Entretanto, quase todos os objetivos e restrição são não lineares, resultando em uma formulação de PNLIM.

Cinco diferentes formas de linearização dos termos não lineares foram considerados para posterior solução das formulações propostas. Nos experimentos realizados considerou-se um conjunto de dados reais oriundo de uma fábrica que realiza operações de corte de metal. Nesse conjunto de dados, considera-se 9 máquinas, agrupadas em três conjuntos diferentes. Desta forma, cada grupo de máquinas idênticas caracteriza uma instância do IPMTC. Os resultados obtidos foram considerados promissores, dado que as soluções ótimas foram obtidas, embora os autores tenham feito ressalvas quanto a aplicação do método em instâncias de maiores dimensões.

3.2 A Definição Formal de Berrada and Stecke (1986)

Posteriormente, Berrada and Stecke (1986) foram os primeiros a relatarem uma definição formal para o IPMTC. Foi considerado um caso particular do problema denominado como Problema de Carregamento em um SMF, cujo objetivo é balancear as cargas de trabalho em cada máquina, representando o quinto problema em um SMF definido por Stecke (1983). O foco dos autores foi ignorar os passos de linearização manual, já que sua aplicação frequente não é justificada, visto que demandam tempo e são pesados, e solucionar os subproblemas não lineares de forma direta e automática. Para tal, em sua PNLIM considera-se os subproblemas de alocação de tarefas às máquinas, o sequenciamento das mesmas e a alocação das ferramentas às máquinas durante o processo. Para solucionar o problema. Define-se então a sequência dos subproblemas e os solucionam com o método *branch-and-bound*, que, aliado à inserção de algumas relaxações ao problema, de forma a minimizar a dificuldade que o problema apresenta por si só, foi capaz de apresentar soluções em tempo computacional curto.

As instâncias consideradas foram divididas em dois conjuntos, contendo máquinas idênticas e máquinas não idênticas, portanto foram mais além do que IPMTC. Um dos conjuntos possui dez instâncias, diferenciadas pelo número de máquinas flexíveis disponíveis, que varia entre 3 e 9, e o número de operações a serem alocadas, que varia entre 7 e 48. O outro conjunto considera quarenta e cinco instâncias, que variam entre 4 e 15 máquinas disponíveis e cujo número de operações varia entre 10 e 40. Concluiu-se que o algoritmo apresentou melhor desempenho quando o tempo de processamento de cada operação é diferente ou quando as máquinas são diferentes. Isso ocorre porque, quando as máquinas ou operações possuem similaridades, o método *branch-and-bound* utilizado gera soluções equivalentes e, por consequência de sua estrutura, não consegue ser tão eficiente ao descartar as soluções não promissoras.

Na maioria das instâncias são considerados tempos de processamento diferentes, variando de acordo com a máquina em que a operação é alocada. Para comparar as instâncias com similaridades nas tarefas ou nas máquinas com instâncias sem similaridades, duas instâncias parecidas, na qual uma possui máquinas não idênticas e a outra considera máquinas idênticas, apresentaram tempos distintos de execução. Para máquinas não idênticas o tempo de execução foi de 0,484 segundos, enquanto que para máquina idênticas o tempo apresentado foi 3,839 segundos.

3.3 O comparativo com *Job Shop Scheduling Problem* de Persi et al. (1999)

A pesquisa realizada por Persi et al. (1999) foi baseada em um caso real de uma fábrica que produz motores a diesel, na Itália, que em sua linha de produção utilizava máquinas paralelas para produzir motores de diversos tamanhos e em lotes com tamanhos diferentes. Assim, é caracterizado algo próximo ao problema em análise neste trabalho, sendo que os motores equivalem às tarefas a serem produzidas no IPMTC e as máquinas, mesmo que não idênticas, se equivalem às máquinas no IPMTC, cujo objetivo é otimizar a eficiência das máquinas.

Com o objetivo de otimizar a utilização das máquinas em um SMF, foi tratado o problema de forma hierárquica, que é o mesmo que propor a divisão do problema em subproblemas hierárquicos. Desta forma, a solução dos subproblemas em níveis hierárquicos menores são incorporadas à solução de subproblemas considerados em hierarquia maior.

1. Divisão das tarefas em lotes;
2. Definição da sequência mais eficaz entres os lotes estabelecidos;
3. Definição a maneira mais eficiente de realizar a transição entre dois lotes; e
4. Definição da ordem das tarefas em cada lotes específico.

No primeiro nível, realiza-se a divisão de todas as tarefas a serem processadas em lotes, observando os lotes que podem ser processados simultaneamente, ou seja, sem a necessidade de realizar troca de ferramentas. O segundo nível define o sequenciamento mais apropriado dos lotes resultantes no primeiro nível, segundo o objetivo geral. O terceiro nível é o que trata da forma de realizar a transição entre dois lotes em uma máquina, de acordo com o sequenciamento definido no nível anterior. Para finalizar, no quarto nível, define-se qual a ordem das tarefas dentro dos lotes já definidos no primeiro nível.

No nível superior da hierarquia, o da divisão de lotes, um modelo de Programação Linear Inteira (PLI) é utilizado, que resulta em minimização no tempo de processamento dos lotes, além de balancear as cargas nas máquinas. O nível de sequenciamento de lotes é modelado como o PCV. Já o planejamento dos lotes é solucionado utilizando três modelos de Programação Inteira Mista (PIM) baseado no *Job Shop Scheduling Problem* (JSSP), que utiliza regras de despacho para solucionar o problema.

As instâncias consideradas são oriundas de casos reais, porém não especificados. Seus resultados não apresentam o valor da solução em si, apesar de serem ótimos, mas consideram o tempo de execução. Os três modelos propostos são equivalentes, já que o objetivo é minimizar o tempo total de produção em todos os modelos. Os modelos são diferenciados por considerarem um planejamento mensal de uma linha de produção, considerarem a produção de uma carga extra e, por ultimo, apresentar um *upper bound* para a produção extra do modelo anterior.

Primeiro conclui, sem evidenciar resultados, que o modelo 2 sempre supera o modelo 1, principalmente para problemas com alto número de tarefas a serem realizadas. Posteriormente, foram consideradas cinquenta e seis instâncias, diferenciadas pelo número de tarefas a serem realizadas, e solucionadas pelo modelo 2. Para qualificar as soluções obtidas, utilizou-se uma formulação matemática que avalia qual a distância da solução até a solução ótima e as analises evidenciou que as soluções estão próximas do ótimo, sendo então boas soluções.

3.4 O Modelo Integrado de Mohamed et al. (1999)

A abordagem feita por Mohamed et al. (1999) busca a minimização do *makespan* e envolve os subproblemas de agrupamento e roteamento (ou sequenciamento) de tarefas, junto ao carregamento de ferramentas na máquina. O primeiro problema em questão envolve a seleção de lotes de tarefas a serem processadas durante um determinado tempo, algo que remete à definição do conjunto de tarefas que será alocada em cada máquina no IPMTC. Já o segundo refere-se a definição do sequenciamento das tarefas em cada máquina, dado o lote destinado à máquina, ou seja, definir a sequência de tarefas que serão processadas por cada máquina, analogamente ao problema de seleção de tarefas a serem processadas em cada uma das máquinas no IPMTC.

Foi proposto o método *Part Grouping, Loading and Routing Model* (PGLRM), considerado um modelo *off-line*, ou seja, todos os dados disponíveis antes de iniciar o planejamento da linha de produção, que se diferencia por resolver o problema como um todo, ao invés de resolver subproblemas para depois unir os resultados gerando a solução final. Desta forma, propôs-se uma divisão igualitária, ou pelo menos balanceada da carga de trabalho, ou seja, as máquinas presentes no SMF possuem carga de trabalho iguais ou parecidas, o que implica em menor tempo de ociosidade da linha de produção. Concluiu-se que o PGLRM gera não apenas menores valores de *makespan*, mas também apresenta uma maior flexibilidade quanto ao roteamento de tarefas.

3.5 A Relação com o Problema do Caixeiro Viajante de Kurz and Askin (2001)

O trabalho de Kurz and Askin (2001) relaciona o clássico Problema do Caixeiro Viajante (PCV) com o IPMTC. O PCV tem como objetivo encontrar a rota entre cidades que possua a menor distância. Como via de regra, o PCV é representado como um grafo completo, onde os vértices representam as cidades, e os pesos das arestas representam a respectiva distância entre duas cidades. Então, em cada máquina no IPMTC, pode-se modelar o sequenciamento de tarefas como uma solução do PCV, em que os vértices representam as tarefas, assim como as cidades no PCV, e as arestas indicam o tempo de *setup* entre duas tarefas, assim como as distâncias entre duas cidades no PCV. Desta forma, foram propostos quatro métodos para solução da modelagem, sendo um algoritmo genético e três heurísticas, que incorporam métodos próprios para o PCV.

No algoritmo genético, o sequenciamento das máquinas é encontrada através do PCV, uma por uma. Posteriormente realiza-se diferentes mutações, que é a troca de alocação de máquina de uma ou mais tarefas, entre os sequenciamentos definidos, afim de encontrar a melhor solução.

A primeira heurística, denominada *Slicing*, adota a ideia de corte na solução da máquina única. Então, constrói uma solução considerando somente uma máquina, através do PCV, e posteriormente a divide em m máquinas.

A heurística *Multiple MULTI-FIT* é o segundo método proposto. Embora a heurística *MULTI-FIT*, incorporada no método composto, não considere tempos de configuração (*setup*) da máquina dependentes da sequência, foi realizada uma integração destes valores com o tempo de processamento, utilizando estimativas (tempo processamento + *setup*). O método então começa executando o *MULTI-FIT* com o tempo modificado, atribuindo tarefas as máquinas. Posteriormente, soluciona-se um PCV em cada máquina para verificar se cada conjunto de tarefas possui o sequenciamento correto, utilizando o cálculo de tempo real. Caso seja determinado um sequenciamento que resulte em um *makespan* melhor, a solução é atualizada.

A terceira heurística apresentada é a *Multiple insertion*, que é uma adaptação da heurística de Inserção própria para o PCV. Assim como o método anterior, este também considera o *setup* incorporado ao tempo de processamento. Aplica-se a heurística de Inserção considerando os tempos modificados de forma crescente, de forma a inserir uma determinada tarefa em todas as possíveis posições da máquina, encontrando assim a melhor *makespan* parcial. Para encontrar o *makespan* real, utiliza o tempo real do sequenciamento e atualiza-o.

160 instâncias foram geradas de forma aleatória, considerando 6 a 100 tarefas e 3 a 10 máquinas disponíveis. Entre os métodos comparados, o *Multiple insertion* apresentou os melhores resultados. O algoritmo genético apresentou bons resultados em alguns casos e, especialmente quanto ao tempo, apresentou-se inferior ao *Multiple insertion*.

3.6 A Definição Atual de Fathi and Barnette (2002)

Uma definição mais concisa e atual do IPMTC, utilizada neste trabalho, foi apresentada por Fathi and Barnette (2002). Adicionalmente, três heurísticas foram propostas, sendo adaptações de heurísticas próprias para outros problemas combinatórios, conforme descrito a seguir.

A primeira heurística proposta utiliza estruturas de vizinhança e estratégias de busca já conhecidas na literatura, como o *Insertion-neighbourhood* e o *Exchange-neighbourhood*. Para tal realiza-se estratégias de vizinhança: busca local de inserção e busca local de troca. O uso dessas estratégias separadas resulta no método *Local Improvement Procedure* (LIP), porém os autores utilizaram o método composto por mais de uma estrutura de vizinhança, o que resulta no método *Composite Local Improve-ment Procedure* (CLIP). Neste trabalho foram propostos dois métodos CLIP, denominados CLIP1 e CLIP2, que se diferenciam pela ordem em que as estruturas de vizinhança são invocados.

O método CLIP1 considera uma solução S inicial, aplica a busca local de inserção para encontrar a solução S', ótima local, e posteriormente aplica-se a busca local de troca na solução S', resultando em uma solução final ótima S''. O método CLIP2 considera também a aplicação da busca local de inserção primeiro, originando a solução S', enquanto que o segundo passo é realizar uma iteração completa da busca local de troca sobre S'. Caso encontre uma solução S'' melhor que S', inicia a busca local de inserção, porém com S'' de solução inicial. Caso contrário, S' é a solução ótima global. Em investigações preliminares, foi constatado que CLIP1 e CLIP2 têm melhor desempenho comparado com as buscas locais isoladamente.

Foram observados elevados tempos de execução em CLIP1 e CLIP2, e para solucionar esse problema foi proposta uma estratégia para reduzir as soluções a serem avaliadas em cada iteração. Aliado a isso, percebeu-se que os múltiplos começos da busca local de troca também consumiam muito tempo, portanto, descarta-se esse passo exploratório e introduz o KTNS, sempre que uma tarefa for adicionada na sequência da máquina.

A segunda heurística se baseia no Problema de Escalonamento em Máquinas Paralelas, uma adaptação da heurística *Longest Processing Time* (LPT), própria para o problema e que utiliza uma lista para realizar a programação das máquinas. A heurística cria uma lista contendo as tarefas, ordenadas pela tarefa com o tempo de processamento mais longo. Quando uma máquina estiver disponível, retira-se uma tarefa da lista e designa à máquina em questão.

A terceira heurística, denominada CSTR, foi proposta observando semelhanças estruturais com o Problema do k-Caixeiro Viajante (k-PCV). Nesta modelagem, cada vértice representa uma tarefas, e cada aresta $\{i, j\}$ representa o sequenciamento da tarefa j após a tarefa i em uma mesma máquina. Para solução do problema, adaptou-se a heurística *Multiple-Start Greedy* (Crama et al., 1994), originalmente projetada para o MTSP, que resolve o sequenciamento em uma única máquina flexível, e posteriormente transforma-se em solução para o IPMTC. Esta

transformação se dá pela divisão da solução anterior em m segmentos, de maneira que cada segmento corresponde ao sequenciamento de tarefas em cada uma das m máquinas. Todas as combinações de divisão de segmentos são analisadas, embora não tenha sido mencionado qual método foi utilizado para tanto.

As instâncias geradas no trabalho foram geradas aleatoriamente, divididas em 3 grupos, utilizando-se diferentes estratégias de construção em cada uma. Estas estratégias foram propostas neste mesmo trabalho, e levam em consideração diferentes critérios para a utilização de ferramentas pelas tarefas, como número máximo de ferramentas, probabilidade de cada ferramenta ser exigida por uma tarefa e estrutura de matriz escada para Q.

Os experimentos computacionais reportaram que CLIP2 obteve melhores soluções, enquanto CSTR apresentou menores tempos computacionais para as instâncias. Analogamente, CLIP1 se manteve estável, com boas soluções e tempos computacionais relativamente baixos, porém piores do que os apresentados em CLIP2.

3.7 O Estado da Arte: Beezão et al. (2017)

Recentemente, Beezão et al. (2017) propuseram duas variações do método *Adaptive Large Neighborhood Search* (ALNS). As variações consideraram diferentes estratégias de solução inicial para o método, sendo uma considerando soluções aleatórias (ALNS-R) e a outra considerando uma heurística própria para geração de soluções para o MTSP desenvolvida por Chaves et al. (2012), denominada ALNS-S.

Os resultados obtidos pelas versões de ALNS foram comparadas entre si e com os métodos CLIP1 e CLIP2, de Fathi and Barnette (2002). Apesar de reportarem somente o *gap* e o tempo de execução entre os quatro métodos comparados e não mostrarem o valor da soluções para futuras comparações, os resultados evidenciaram que ALNS-R possui melhor desempenho, apresentando as melhores soluções em tempos de execução consideravelmente baixos, porém, maiores que os reportados por CLIP2, sendo então considerado o estado da arte para soluções do IPMTC.

Para ilustrar o desempenho dos métodos nas soluções IPTMC-I e IPTMC-II, a Tabela 3.1 é extraída do trabalho. A tabela mostra os valores de *gap*, desvio percentual médio das soluções, e *gap**, desvio percentual médio em relação à melhor solução encontrada, embora não especificada.

Tabela 3.1: *Gap* obtidos em IPTMC-I e IPTMC-II.

	CLIP1		CLIP2		ALNS-B		ALNS-R	
	gap	*gap**	*gap*	*gap**	*gap*	*gap**	*gap*	*gap**
IPTMC-I	17,90	7,40	15,76	5,76	0.18	0.02	0.18	0.02
IPTMC-II	18,16	8,76	14,66	7,08	–	–	2,13	0.12

Para o conjunto IPTMC-II, ALNS-B apresentou dificuldades, mesmo em casos com 50 tarefas. Para as instâncias com maior número de tarefas (100 e 200) o método atingiu cerca de 3600 segundos antes de terminar a busca da solução inicial. Já para instâncias com 50 tarefas, o algoritmo foi completamente executado, porém, em média, mais de 80% do tempo de execução foi gasto na busca da solução inicial.

Capítulo 4

Fundamentação Teórica

Neste capítulo é apresentado o embasamento teórico para a implementação dos métodos. O *ILS* proposto por Paiva and Carvalho (2017), atual estado da arte para o MTSP, é o alvo de estudos para resolver problemas similares ao estudado por eles. Para entendimento dos métodos apresentados a seguir, considere que o plano de trocas de ferramentas de uma máquina é representado por uma matriz binária M^{ϕ}, sendo que as colunas da são sequenciadas por ϕ e o valor de cada elemento $m^{\phi}_{i,j}$ é igual a 1 caso a ferramenta i esteja na máquina durante a execução da tarefa j. Caso contrario $m^{\phi}_{i,j}$ vale 0.

4.1 *1-block* de Crama et al. (1994)

Em sua definição, *1-block* é um padrão encontrado nas soluções para o MTSP, que, dado uma matriz binária do MTSP, analisa iterativamente cada uma de suas linhas, afim de encontrar um conjunto de elementos com valor igual a 1 consecutivos. Assim, cada *1-block* evidencia uma troca de ferramentas, já que uma sequencia dentro da linha da matriz assumira o mesmo valor 1. Como a função de avaliação do MTSP considera o número de inversões de 0 para 1 em cada linha da matriz, o número de trocas de ferramentas, dado uma sequencia ϕ, é igual ao número de *1-blocks*, encontrados na matriz M^{ϕ}.

4.2 O trabalho de Paiva and Carvalho (2017)

O atual estado da arte para o MTSP, Paiva and Carvalho (2017) propuseram uma Busca Local Iterada, ou *Iterated Local Search* (ILS), unindo métodos para a sua implementação. Foram utilizados três métodos, a modificação na representação do MTSP por grafos, uma nova heurística e um novo método de busca local.

4.3 Representação do MTSP por Grafos

A nova representação do problema por grafos é representada por um *Grafo de Ferramentas* (Paiva and Carvalho, 2017), e é definido por $G = (V, E)$. O conjunto de vértices V, representa as ferramentas, já o conjunto de arestas E, indica se a ferramenta i é utilizada ao mesmo tempo que a ferramenta j. Então o peso de cada aresta indica a frequência em que as ferramentas i e j são necessárias em uma mesma tarefa.

4.4 Solução Inicial

Assim como Paiva and Carvalho (2017), este trabalho adota a execução do método Busca em Largura (ou *Breadth-First Search* – BFS) para determinar o sequenciamento ϕ das tarefas, que indica a ordem das colunas da matriz M^{ϕ} para execução do ILS. Dado um Grafo de Ferramentas, i.e, identificação de quais ferramentas tem mais frequência de utilização, a BFS inicia sua execução a partir de um vértice juntamente com a adição do critério, baseado no peso das arestas do grafo. Assim, a probabilidade de encontrar *1-blocks* é maior quando as ferramentas com maior peso nas arestas estiverem próximas no sequenciamento de tarefas. Então, o critério de maior peso das arestas é o guia da Busca em Largura. O resultado é o sequenciamento F_{ϕ} de ferramentas.

Algoritmo 1: BFS

```
   Entrada: G = (V, E)
1  início
2      S_φ ← ∅;
3      Fila F ← ∅;
4      Defina vértice inicial v;
5      S_φ ← v;
6      enquanto Existir vértice em V não visitado faça
7          enquanto F ≠ ∅ faça
8              x ← remove elemento de F;
9              Marque x como visitado;
10             vizinhana ← ordenação decrescente dos vizinhos de x , de acordo com o peso
                das arestas;
11             para cada y ∈ vizinhana faça
12                 se x ∉ S_φ então
13                     F ← x;
14                     S_φ ← x;
15                 fim
16             fim
17         fim
18         se existir um vértice k ∈ V ainda não visitado então
19             F ← k;
20         fim
21     fim
22     retorna F_φ
23 fim
```

Dado o sequenciamento de ferramentas F_{ϕ} resultante da execução do algoritmo BFS, o sequenciamento ϕ é definido seguindo os passos a seguir. Percorre o sequenciamento $F\phi$, sendo que em cada iteração k a ferramenta na posição k fica disponível. Verifica-se se o conjunto de ferramentas disponíveis é capaz de executar uma tarefa, e se for possível, insere a tarefa em ϕ. Caso contrário, continua a próxima iteração de k até percorrer todo o sequenciamento F_{ϕ}. Caso duas ou mais tarefas tornarem aptas à execução ao mesmo tempo, o critério de desempate é a tarefa com maior número de ferramentas. O sequenciamento ϕ de tarefas é o dado de entrada do ILS.

No algoritmo a seguir considere F_{disp} o conjunto de ferramentas disponíveis e T_{aptas} conjunto de tarefas que estão aptas a serem executadas.

Algoritmo 2: Sequenciamento de tarefas

Entrada: F_ϕ
Saída: Sequenciamento de tarefas ϕ

início
  $F_{disp} \leftarrow \emptyset$;
  $\phi \leftarrow \emptyset$;
  para cada $x \in F_\phi$ **faça**
    $F_{disp} \leftarrow F_{disp} \cup x$;
    $T_{aptas} \leftarrow \{i \in T \mid F_i \subseteq F_{disp}$ e $i \notin \phi\}$;
    enquanto $T_{aptas} \neq \emptyset$ **faça**
      se $\phi = \emptyset$ **então**
        $k \leftarrow$ tarefa $j \in T_{aptas}$ com o maior $\mid F_j \mid$;
      fim
      senão
        $k \leftarrow$ tarefa $j \in T_{aptas}$ que minimize $Z^{MTSP}(M^{\phi \cup j})$;
      fim
      Remove k de T_{aptas};
      $\phi \leftarrow k$;
    fim
  fim
  retorna ϕ
fim

4.5 Busca Local para Agrupamento de *1-blocks*

A estratégia utilizada foi a busca por dois ou mais *1-blocks* na mesma linha da matriz M^ϕ. Caso encontre *1-blocks* na linha, tenta-se agrupá-los sucessivamente dois a dois, movimentando cada uma das colunas relacionadas ao primeiro *1-block* para antes ou depois do segundo *1-block* encontrado, considerando sempre o melhor movimento. Caso todos os movimentos realizados piorem a solução, o movimento não é efetivado.

Algoritmo 3: Agrupamento *1-block*

Entrada: Matriz M^ϕ
Saída: Matriz M^ϕ modificada

início
  para cada *linha de* M^ϕ **faça**
    encontre o primeiro *1-block* i;
    enquanto *existir mais 1-block j* **faça**
      Para cada coluna de i, decida se esta deve ser inserida antes ou depois de j ou ser mantida em i;
      $i \leftarrow j$;
    fim
  fim
  retorna S
fim

4.6 Busca Local Iterada

O método ILS tem como objetivo diminuir o número de iterações apresentadas no método de Crama et al. (1994) pela diminuição do número de *1-blocks* presentes em cada linha da matriz de permutação no MTSP.

Capítulo 5

Desenvolvimento

Nesta seção é apresentando os métodos implementados neste trabalho, evidenciando as estruturas fundamentais para as suas respectivas construções. Para facilitar a descrição dos métodos considere m o número de máquinas disponíveis e n o número de tarefas a serem realizadas.

5.1 Representação de uma Solução

De acordo com o proposito inicial do trabalho, deve-se atribuir tarefas às máquinas disponíveis, isto é, uma solução é representada pela subdivisão de n tarefas em m máquinas. Sendo assim, uma solução é particionada em subconjuntos, em que cada subconjunto indica quais tarefas serão alocadas as máquinas. Seja $N = \{1, \ldots, n\}$ o índice de todas tarefas e $S = \{N^1, \ldots, N^m\}$ representando a divisão de N em m subconjuntos, onde N^i indica o subconjunto alocado para a máquina i.

5.2 Soluções iniciais

Para construção das soluções iniciais utilizadas neste trabalho, é necessário o entendimento do conceito de lista de processamento, muito comum em alguns procedimentos heurísticos para o Problema da Máquina Paralela. A ideia é construir uma lista de tarefas, e posteriormente subdividi-la, como várias listas definitivas. A lógica ao preencher as listas é: sempre que uma máquina estiver disponível, aloque uma tarefa à ela. Uma subjetividade quanto a escolha da tarefa a ser inserida é evidenciada. Então, para escolha da tarefa foram adotados o procedimento Tempo de Processamento Mais Longo, do inglês *Longest Processing Time* (LPT), e o Procedimento Aleatório de Processamento de Lista, também do inglês *Random List Processing Procedure* (RLPP).

5.2.1 *Longest Processing Time* (LPT)

O procedimento LPT constrói a lista considerando só, e somente só, os tempos de processamento das tarefas, não se importando assim com o tempo gasto para as trocas de ferramentas. As tarefas são sequenciadas de forma crescente, considerando o tempo de processamento, seleciona a tarefa com o maior tempo de processamento ainda não alocada e insere na lista da máquina com o menor tempo de processamento total, e assim consecutivamente até todas as tarefas serem alocadas.

Para o pseudo código a seguir, consideramos as seguintes funções:

- *tarefaMaiorTempo*(ϕ) - retorna tarefa com maior tempo de processamento do conjunto ϕ;
- *maquinaMenorTempo*(ϕ) - retorna o índice da máquina, com o menor tempo total de processamento das tarefas alocadas a ela, do conjunto ϕ;
- *Custo*(ϕ,k) - retorna o custo de inserir a tarefa k no fim do sequenciamento ϕ;
- *numerodeTrocas*(ϕ) - retorna o número de trocas de ferramentas do sequenciamento ϕ.

Algoritmo 4: LPT

Entrada: Conjunto N, contendo n tarefas
Saída: Sequenciamento S, contendo m sequenciamentos de tarefas

```
1  início
2      para cada m' ∈ S faça
3          S(m')(tempo) ← 0;
4      fim
5      enquanto N ≠ ∅ faça
6          t ← tarefaMaiorTempo(N);
7          m' ← maquinaMenorTempo(S);
8          S(m') ← S(m') ∪ {t};
9          S(m')(tempo) ← S(m')(tempo) + t(tempo);
10         N ← N \ {t};
11     fim
12     retorna S
13 fim
```

5.2.2 *Random List Processing Procedure* (RLPP)

O procedimento RLPP adota o critério de aleatoriedade para geração de suas listas. Para cada instância são geradas várias listas, aplicando a heurística de procedimento de lista em cada uma. Posteriormente aplica-se o método *Multiple-Start Greedy* (MSG) (Crama et al., 1994) em cada uma das listas para selecionar o melhor sequenciamento dentro do conjunto de tarefas selecionado. Assim com Beezão et al. (2017), limitamos o número de listas aleatórias geradas por instância em 50.

A heurística MSG é eficaz, simples e possui o seguinte objetivo: a partir de uma tarefa inicial, insere-se tarefas, uma a uma e seguindo um critério de avaliação, afim de formar o melhor sequenciamento dentre as tarefas do conjunto. O critério de avaliação é politica ótima KTNS (Tang and Denardo, 1988), de forma que, dado uma sequencia de tarefas e as tarefas a serem inseridas, seleciona a tarefa que ao ser inserida no fim do sequenciamento resulte no menor número de troca de ferramentas, e assim consecutivamente até não existirem tarefas a serem inseridas. A execução da heurística considera cada uma das tarefas do conjunto como a tarefa inicial.

Algoritmo 5: *MSG*

Entrada: Conjunto T de tarefas
Saída: Melhor sequenciamento do conjunto T

início
  $melhorSequencia \leftarrow \emptyset$;
  para cada $t \in T$ **faça**
    $sequencia \leftarrow \{t\}$;
    $T \leftarrow T \backslash \{t\}$;
    enquanto $T \neq \emptyset$ **faça**
      $t \leftarrow \min(Custo(sequencia, j)), j \in T$;
      $sequencia \leftarrow sequencia \cup \{t\}$;
      $T \leftarrow T \backslash \{t\}$;
    fim
    se $numerodeTrocas(sequencia) < numerodeTrocas(melhorSequencia)$ **então**
      $melhorSequencia \leftarrow sequencia$;
    fim
  fim
  retorna *melhorSequencia*
fim

Algoritmo 6: *RLPP*

Entrada: Conjunto T de tarefas
Saída: Sequenciamento S do conjunto T

início
  $S \leftarrow \emptyset$;
  $makespan \leftarrow \infty$;
  para *1:50* **faça**
    $aleatorio \leftarrow random(T)$;
    para cada $t \in aleatorio$ **faça**
      $m' \leftarrow maquinaMenorTempo(S)$;
      $S(m') \leftarrow S(m') \cup \{t\}$;
      $S(m')(tempo) = S(m')(tempo) + t(tempo)$;
    fim
    $maux \leftarrow \infty$;
    para $i = 1 : m$ **faça**
      $auxiliar(i) \leftarrow MSG(S(i))$;
      if $numeroTrocas(auxiliar(i)) < maux$ **then**
        $maux \leftarrow numeroTrocas(auxiliar(i))$;
    fim
    se $maux < makespan$ **então**
      $S \leftarrow auxiliar$;
      $makespan \leftarrow maux$;
    fim
  fim
  retorna *Sequenciamento*
fim

5.3 Estrutura de Vizinhança

Para entendimento das estruturas de vizinhança que serão apresentadas a seguir são necessárias as seguintes definições: $i*$ representa a máquina critica da solução S, ou seja, aquela o *makespan* da solução é representado pelo seu tempo de processamento, N^{i^*} o conjunto de tarefas representado pela máquina crítica.

São definidas duas estruturas de vizinhança:

- *Vizinhança de Inserção*: Uma solução S' é dita vizinho de inserção, se for obtido de S, removendo uma tarefa t de N^{i^*}, e inserindo-a em um conjunto N^i $(i \neq i^*)$. Desta forma, dado um conjunto S, o tamanho de sua vizinhança de inserção é igual a $\mid N^{i^*} \mid *(m-1)$, onde $\mid N^{i^*} \mid$ representa o número de tarefas em N^{i^*};

- *Vizinhança de Troca*: Uma solução S' é dita vizinho de troca, se for obtido de S, realizando a troca de alocação de máquina da tarefa $t_1 \in N^{i^*}$ com a tarefa $t_2 \in N^i (i \neq i*)$, ou seja, remoção das tarefas $t_1 \in N^{i^*}$ e $t_2 \in N^i$ $(i \neq i^*)$ e inserção das mesmas tarefas t_1 em N^i e t_2 em N^{i^*} . Desta forma, dado um conjunto S, o tamanho de sua vizinhança de troca é igual a $\mid N^{i^*} \mid *(n- \mid N^{i^*} \mid)$.

Para limitar o espaço e o tempo de busca, foi utilizado um critério de porcentagem especifica, considerando a tarefa e o conjunto de tarefas que a tarefa requer a sua inserção, para verificar uma estrutura de vizinhança válida. Para vizinhança de inserção só, e somente só, é considerado uma vizinhança válida em se pelo menos 50% da ferramentas necessárias para o processamento da tarefa t forem exigidas para processamento de todo o conjunto em que ela pleiteia ser inserida. Da mesma forma, na vizinhança de troca, a vizinhança é válida se, e somente se, ambas as tarefas tiverem pelo menos 50% das ferramentas em comum com o conjunto a serem inseridas, respectivamente. A porcentagem foi determinada por Fathi and Barnette (2002), através de testes preliminares.

5.4 Estrategias de busca

As estratégias de busca utilizadas neste trabalho foram: *primeira melhoria* e *melhor melhoria*. Foram utilizadas nas implementações a estratégia primeira melhoria na Vizinhança de Troca, denominado por nos de "e-rotina", e uma estrategia mista na Vizinhança de Inserção, a "i-rotina".

Também em testes preliminares, Fathi and Barnette (2002) constataram que na estratégia "e-rotina"é necessário muito tempo para a aplicação repetida da heurística MSG. Então, após remover a tarefa t_2 da máquina crítica (N^{i^*}), insere a tarefa t_2 em todas as posições da máquina N^i e aplica KTNS em todas as inserções. Analogamente realiza o mesmo procedimento, porém retirando a tarefa t_1 da máquina N^i e inserindo em todas as posições da máquina crítica (N^{i^*}). Então o movimento é comparado com o sequenciamento corrente, através do *makespan*, e atualiza a solução caso for melhor.

Para a estratégia mista nos consideramos as máquinas da solução S em ordem crescente de processamento das tarefas, então $P^1 \leq P^2 \leq \ldots \leq P^m$, claramente P^m representa a máquina crítica. Iniciamos a "i-rotina"removendo cada tarefa $t \in N^{i^*}$ e inserindo na máquina N^1, uma por vez, e a cada iteração aplicamos a heurística MSG para obter os *makespan* correspondentes em cada máquina, N^1 e N^{i^*}. Então seleciona a inserção que resultou no menor *makespan* e o compara com o *makspan* corrente. Se o *makespan* selecionado for menor que o *makespan* corrente, o novo sequenciamento torna o sequenciamento atual, por consequência atualiza o

makespan corrente, e começa de novo. Caso contrário, consideramos a próxima máquina da ordem (P^2) e continua a operação. O método chega ao fim quando não melhorar a solução ou não ser mais possível construir soluções, ou seja, depois da análise todas as máquina e não houver mudanças.

5.5 Procedimentos Compostos

Cada uma das estruturas apresentadas podem ser utilizadas para criação de um Procedimento de Melhoria Local, do inglês *Local Improvement Procedure* (LIP), sendo que os implementados nesse trabalho foram denominados "i-rotina"e "e-rotina". Existe métodos utilizam diferentes estruturas de vizinhança alternando a sua ordem, estes são denominados Procedimento de Melhoria Local Composto (*Composite Local Improvement Procedures* - CLIP). Para esse trabalho foi implementado os métodos $CLIP_1$ e $CLIP_2$, assim como Fathi and Barnette (2002).

- $CLIP_1$: Inicializa com uma solução S, realiza a "i-rotina"e encontrando uma solução S' (S pode ser S'). Então aplica a "e-rotina"em S' enquanto o resultado for melhorando, encontrando assim o ótimo global;
- $CLIP_2$: Inicializa com uma solução S, realiza a "i-rotina"e encontrando uma solução S' (S pode ser S'). Então inicializa o procedimento completo "e-rotina"em S' encontrando S'', ou seja, realiza reinicializa "e-rotina"sempre ela melhorar a solução. Se S'' for melhor que S', então reinicializa "i-rotina"com S'' de solução inicial. Caso contrário termina o procedimento com S'.

Algoritmo 7: $CLIP_1$

```
   Entrada: Sequenciamento S
1  início
2      S' ← i − rotina(S);
3      enquanto teste = false faça
4          S'' ← e − rotina(S');
5          se makespan(S'') < makespan(S') então
6              S' ← S'';
7              teste ← true;
8          fim
9          senão
10             teste ← false;
11         fim
12     fim
13     retorna S'
14 fim
```

Algoritmo 8: $CLIP_2$

Entrada: Sequenciamento S

início
  enquanto $teste = false$ **faça**
    $S' \leftarrow i - rotina(S)$;
    $S'' \leftarrow e - rotina(S')$;
    se $makespan(S'') < makespan(S')$ **então**
      $S \leftarrow S''$;
      $teste \leftarrow true$;
    fim
    senão
      $teste \leftarrow false$;
    fim
  fim
  retorna S'
fim

Capítulo 6

Resultados e Discussão

Beezão et al. (2017) geraram dois conjuntos de instâncias, IPMTC-I e IPMTC-II, baseados nas instâncias do MTSP de Yanasse et al. (2009) e Crama et al. (1994). As características destas instâncias são apresentadas na Tabela 6.2. O conjunto IPMTC-I (*a*) apresenta 1440 instâncias, que se subdividem primeiro pelo número de máquinas (m), depois pelo número de tarefas (n) e, por último, pelo número de ferramentas (f). Também com 1440 instâncias, o conjunto IPMTC-II (*b*) se subdivide primeiro por n, depois por m, e por último, por f. Também na tabela é apresentado o número total de instâncias contidas em cada um dos subconjuntos descritos.

Tabela 6.1: Conjunto IPMTC-I.

m	n	f	Total
2	{8,15,25}	{15,20}	720
3	{15,25}	{15,20}	480
4	{25}	{15,20}	240

Tabela 6.2: Conjunto IPMTC-II.

m	n	f	Total
{3,4,5}	50	{30,40}	360
{4,5,6,7}	100	{30,40}	480
{6,7,8,9,10}	200	{30,40}	600

Quatro métodos foram implementados e testados sobre IPMTC-I e IPMTC-II. São eles: *CLIP*$_1$*-RLPP*, *CLIP*$_2$*-RLPP*, *CLIP*$_1$*-LPT* e *CLIP*$_2$*-LPT*:

- *CLIP*$_1$*-RLPP* e *CLIP*$_2$*-RLPP* utilizaram o método *RLPP* para construção das soluções iniciais;
- *CLIP*$_1$*-LPT* e *CLIP*$_2$*-LPT* utilizaram o *LPT* para geração de soluções iniciais.

Os métodos foram implementados utilizando a linguagem C++. Os experimentos foram realizados em um computador *Intel(R) Core(TM)* i7-4790 CPU @ 3.60GHz, 16 GB de RAM e sistema operacional Ubuntu 16.04.02. Como *RLPP* adota aleatoriedade para construção das soluções, foram realizadas 10 execuções diferentes em cada instância por método. O mesmo não é verificado em LPT, já que a solução gerada pelo método é sempre a mesma.

Seguindo o padrão das tabelas apresentadas por Beezão et al. (2017), nas tabelas a seguir são apresentados o número de máquinas (m), o número de tarefas (n), o número de ferramentas

(f). Para soluções encontradas em $CLIP_1$-*RLPP* e $CLIP_2$-*RLPP*, é apresentado a média das melhores soluções encontradas em cada instância do conjunto (S*), a média de todas as soluções encontradas para cada instância (S), o desvio padrão (σ) e o tempo (t). Já em $CLIP_1$-*LPT* e $CLIP_2$-*LPT* temos a solução encontrada pelo método (S) e o tempo (t).

Tabela 6.3: IPMTC-I.

m	n	f	$CLIP_1$-*RLPP*				$CLIP_2$-*RLPP*				$CLIP_1$-*LPT*		$CLIP_2$-*LPT*	
			S*	S	σ	t	S*	S	σ	t	S	t	S	t
2	8	15	457,93	480,37	18,45	0,04	457,87	480,18	18,53	0,39	504,81	0,00	504,29	0,00
2	8	20	675,75	705,53	21,72	0,05	675,56	705,36	21,77	0,51	730,55	0,00	727,71	0,00
2	15	15	652,09	686,78	27,09	0,53	652,03	686,82	27,35	0,54	746,50	0,17	748,76	0,18
2	15	20	802,61	834,35	22,21	0,53	802,35	834,68	23,13	0,54	874,45	0,15	875,43	0,16
2	25	15	804,82	832,92	19,42	3,49	804,45	833,23	20,26	3,61	895,97	2,58	896,04	2,65
2	25	20	931,41	968,66	25,49	5,43	930,83	969,50	26,67	5,60	1054,10	4,29	1054,10	4,39
3	15	15	512,69	543,52	21,35	0,15	512,04	543,22	21,38	0,23	567,76	0,03	566,82	0,03
3	15	20	618,88	642,93	16,69	0,15	618,23	642,26	17,13	0,30	654,36	0,02	653,68	0,03
3	25	15	634,43	661,55	15,97	0,86	633,86	660,50	15,90	0,88	673,88	0,28	673,84	0,31
3	25	20	761,48	790,90	17,63	1,27	760,01	789,97	17,85	1,31	810,40	0,43	809,92	0,48
4	25	15	510,88	532,90	13,53	0,33	510,18	531,77	13,26	0,34	537,78	0,08	536,33	0,09
4	25	20	640,88	667,15	15,64	0,47	640,18	665,71	15,3	0,49	677,34	0,12	675,78	0,13

Tabela 6.4: IPMTC-II.

m	n	f	$CLIP_1$-$RLPP$				$CLIP_2$-$RLPP$				$CLIP_1$-LPT		$CLIP_2$-LPT	
			S^*	S	σ	t	S^*	S	σ	t	S	t	S	t
3	50	30												
3	50	40												
4	50	30												
4	50	40												
4	100	30												
4	100	40												
5	50	30												
5	50	40												
5	100	30												
5	100	40												
6	100	30												
6	100	40												
6	200	30												
6	200	40												
7	100	30												
7	100	40												
7	200	30												
7	200	40												
8	200	30												
8	200	40												
9	200	30												
9	200	40												
10	200	30												
10	200	40												

Capítulo 7

Conclusões

Referências Bibliográficas

Andreza Cristina Beezão, Jean-François Cordeau, Gilbert Laporte, and Horacio Hideki Yanasse. Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, 257(3):834–844, 2017.

Mohammed Berrada and Kathryn E Stecke. A branch and bound approach for machine load balancing in flexible manufacturing systems. *Management Science*, 32(10):1316–1335, 1986.

Antonio Augusto Chaves, Edson Luiz França Senne, and Horacio Hideki Yanasse. Uma nova heurística para o problema de minimização de trocas de ferramentas. *Gestão & Produção*, pages 17–30, 2012.

Yves Crama, Antoon W. J. Kolen, Alwin G. Oerlemans, and Frits C. R. Spieksma. Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6(1):33–54, 1994. ISSN 1572-9370. doi: 10.1007/BF01324874. URL http://dx.doi.org/10.1007/BF01324874.

Y Fathi and KW Barnette. Heuristic procedures for the parallel machine problem with tool switches. *International Journal of Production Research*, 40(1):151–164, 2002.

Michael R Garey and David S Johnson. *Computers and intractability*. Freeman, 1979.

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

Zubair M Mohamed, Ashok Kumar, and Jaideep Motwani. An improved part grouping model for minimizing makespan in fms. *European journal of operational research*, 116(1):171–182, 1999.

Gustavo Silva Paiva and Marco Antonio M. Carvalho. Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers Operations Research*, 88:208 – 219, 2017. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2017.07.013. URL http://www.sciencedirect.com/science/article/pii/S0305054817301971.

Piero Persi, Walter Ukovich, Raffaele Pesenti, and Marino Nicolich. A hierarchic approach to production planning and scheduling of a flexible manufacturing system. *Robotics and Computer-Integrated Manufacturing*, 15(5):373–385, 1999.

Kathryn E Stecke. Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, 29(3):273–288, 1983.

Kathryn E Stecke and James J Solberg. Loading and control policies for a flexible manufacturing system. *The International Journal of Production Research*, 19(5):481–490, 1981.

Christopher S Tang and Eric V Denardo. Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. *Operations research*, 36(5):767–777, 1988.

Horacio Hideki Yanasse, Rita de Cássia Meneses Rodrigues, and Edson Luiz França Senne. Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. *Gestão and Produção*, 13(3), 2009.