Leonardo Cabral da Rocha Soares

Otimização de processos produtivos em sistemas de manufatura flexível

Ouro Preto

2021

Leonardo Cabral da Rocha Soares

Otimização de processos produtivos em sistemas de manufatura flexível

Tese submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto para o exame de qualificação de doutorado em Ciência da Computação

Universidade Federal de Ouro Preto - UFOP

Departamento de Computação

Programa de Pós-Graduação em Ciência da Computação

Orientador: Marco Antonio Moreira de Carvalho

Ouro Preto

2021

Leonardo Cabral da Rocha Soares
Otimização de processos produtivos em sistemas de manufatura flexível/ Leonardo Cabral da Rocha Soares. – Ouro Preto, 2021-
128 p. : il. (algumas color.) ; 30 cm.

Orientador: Marco Antonio Moreira de Carvalho

Tese (Doutorado) – Universidade Federal de Ouro Preto - UFOP
Departamento de Computação
Programa de Pós-Graduação em Ciência da Computação, 2021.

1. Escalonamento. 2. Máquinas Flexíveis. 2. Metaheurística. I. Carvalho, Marco Antonio Moreira de. II. Universidade Federal de Ouro Preto. III. Departamento de Computação. IV. Otimização de processos produtivos em sistemas de manufatura flexível.

Leonardo Cabral da Rocha Soares

Otimização de processos produtivos em sistemas de manufatura flexível

Tese submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto para o exame de qualificação de doutorado em Ciência da Computação

Ouro Preto, 12 de julho de 2021.

Marco Antonio Moreira de Carvalho
Orientador

Antônio Augusto Chaves
Membro Externo

Marcone Jamilson Freitas Souza
Membro Interno

Túlio Ângelo Machado Toffolo
Membro Interno

Ouro Preto
2021

Resumo

Sistemas de manufatura flexíveis (SMF) estão presentes em diversos segmentos industriais desde a década de 1960. A evolução dos sistemas computadorizados de controle permitiu uma maior flexibilização no *mix* e volume de peças produzidas. Já na década de 1980, estudos apontavam para um grande crescimento dos SMFs. Ao se perceber a complexidade geral do gerenciamento de produção de um SMF, diversos autores dedicaram-se ao estudo dos problemas computacionais advindos de tal modelo de produção. Estudos recentes demonstram que o número de publicações em temas correlatos cresce constantemente desde o ano de 1988, reforçando a relevância e atualidade do tema. Neste trabalho, são abordados alguns dos principais problemas advindos deste cenário. Inicialmente, o problema de minimização de trocas de ferramentas e o problema do escalonamento de tarefas em máquinas paralelas idênticas com restrições de ferramentas são apresentados para fornecer o referencial teórico que serve de base para o estudo dos demais problemas abordados. Destes dois problemas derivam diversos outros problemas combinatoriais cujo estudo pode ser diretamente aplicado ao setor industrial, além de contribuir para a comunidade acadêmica pois tratam-se, em sua maioria, de problemas $\mathcal{NP}$-difíceis. Em seguida, são abordados o problema de minimização de blocos de uns consecutivos e o problema de sequenciamento de tarefas em máquinas paralelas com limitação de recursos. Para ambos os problemas, são apresentados uma ampla revisão bibliográfica e propostos métodos para solução computacional. Os experimentos computacionais consideram os conjuntos *benchmark* de instâncias disponíveis e os resultados reportados superam os existentes na literatura. Por fim, são apresentadas as conclusões e o plano de atividades restantes para conclusão do programa de doutorado.

Palavras-chave: Escalonamento, Manufatura Flexível, Problemas combinatoriais, metaheurísticas.

Abstract

Flexible manufacturing systems (FMS) have been present in several industrial segments since the 1960s. The evolution of computerized control systems has allowed greater flexibility in the mix and volume of parts produced. Since the 1980s, studies pointed to the great growth of FMS. When realizing the general complexity of the production management of an FMS, several authors dedicated themselves to the study of the computational problems arising from such production model. Recent studies demonstrate that the number of publications on related topics has grown steadily since 1988, reinforcing the relevance and topicality of the topic. In this thesis, some of the main problems arising from this scenario are addressed. Initially, the job sequencing and tool switching problem and the problem of identical parallel machines with tooling constraints are presented to provide the theoretical framework that serves as basis for the study of the other problems discussed. From these problems, several other combinatorial problems branch, whose study can be directly applied to the industrial sector in addition to contributing to the academic community, as they are mostly $\mathcal{NP}$-hard problems. Next, the problem of consecutive block minimization and the problem of resource-constrained parallel machine scheduling are addressed. For both problems, extensive literature reviews and methods for computational approaches are presented. The computational experiments consider publicly available benchmark instances, and the reported results outperform the best known results from the literature. Finally, the conclusions and the plan of remaining activities for completion of the doctoral program are presented.

Keywords: Scheduling, Flexible Manufacturing, Combinatorial problems, metaheuristics.

Declaração

Este documento é resultado de meu próprio trabalho, exceto onde referência explícita é feita ao trabalho de outros, e não foi submetida para outra qualificação nesta nem em outra universidade.

Leonardo Cabral da Rocha Soares

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

2-TSP	*Traveling Salesman Problem of Second Order*
AG	Algoritmo Genético
ALNS	*Adaptative Large Neighborhood Search*
BFS	*Breadth-first Search*
BRKGA	*Biased Random-key Genetic Algorithm*
CBM	*Consecutive Block Minimization*
CIR	*Circular block Minimization*
CLIP	*Composite Local Improvement Procedure*
CS	*Clustering Search*
CSP	*Cutting-stock Problem*
CSTR	*Constructive Heuristic*
DPSO	*Discrete Particle Swarm Optimal*
GMLP	*Gate Matrix Layout Problem*
GTSP	*Generalized Traveling Salesman Problem*
HGS	*Hybrid Generic Search*
HTSP	*Hamming Traveling Salesman Person*
ILS	*Iterated Local Search*
IPMTC	*Identical Parallel Machines With Tooling Constraints*
KTNS	*Keep Tool Needed Soonest*
LIP	*Local Improvement Procedure*
LPT	*Longest Processing Time*
MSG	*Multiple-Start Greedy*
RKGA	*Random-key Genetic Algorithms*

RLPP	*Random List Processing Procedure*
RCPMS	*Resource Constrained Parallel Machine Scheduling*
SMF	Sistema de Manufatura Flexível
SPM	*Scheduling Problem Of Parallel Machines*
SSP	*Job Sequencing and tool Switching Problem*
TSP	*Traveling Salesman Problem*
VND	*Variable Neighborhood Descent*
VNS	*Variable Neighborhood Search*

Lista de símbolos

Δ	*Makespan*
Δ_m	Tempo de processamento da máquina m
J	Conjunto de tarefas
M	Conjunto de máquinas flexíveis
P	População em algoritmos genéticos
$\bar{p}$	Tempo para troca de uma ferramenta
T	Conjunto de ferramentas

Sumário

1 Introdução

De forma geral, sistemas de manufatura podem ser definidos como um conjunto de equipamentos e recursos integrados com objetivo de realizar operações de processamento ou montagem em matéria-prima ou produtos semiacabados (MIKELL, 2011). O conjunto de equipamentos e recursos utilizados é composto por máquinas e ferramentas de produção, dispositivos para manuseio de material e posicionamento de trabalho, além de sistemas de computação. De forma genérica, as operações executadas por máquinas pertencentes a um sistema de manufatura sobre matéria-prima ou produtos semiacabados são denominadas como *processamento de tarefas*. De forma análoga, o resultado obtido a partir do processamento de uma tarefa será referido neste trabalho como *peça*, podendo a mesma representar um produto acabado ou apenas uma operação intermediária no processo produtivo.

Durante a década de 1960, a competição pelo mercado de produtos manufaturados tornou-se mais complexa (SHIVANAND, 2006). Custo de produção, qualidade dos produtos e velocidade na entrega variavam como prioridade a ser considerada. Frente a este novo cenário, as empresas precisavam adaptar seus sistemas de produção visando maior agilidade e flexibilidade. A Figura 1, adaptada de Shivanand (2006), ilustra exigências do mercado e possíveis respostas dos sistemas de manufatura. Para viabilizar a flexibilização dos produtos entregues, o sistema de manufatura deve ser capaz de flexibilizar o volume processado. Para diversificar as peças produzidas, o sistema de manufatura deve ser flexível.

Para que a flexibilização dos sistemas de manufatura fosse possível, fez-se necessário o desenvolvimento prévio de máquinas de controle numérico (CN). De acordo com Thyer (2014), as primeiras máquinas de controle numérico foram desenvolvidas entre 1947 e 1952 no *Massachusetts Institute of Technology*, em conjunto com *Parsons Aircraft Corporation*. As primeiras máquinas utilizavam o controle numérico no posicionamento do trabalho em relação à ferramenta, não contendo suporte para troca automática de ferramentas. O avanço da tecnologia e o aumento da capacidade de controle com os controles numéricos computadorizados (CNCs), permitiu a adaptação destas máquinas com o trocador automático de ferramentas, o que foi fundamental para o desenvolvimento dos Sistemas de Manufatura Flexíveis.

Sistemas de Manufatura Flexíveis (SMFs) são constituídos por uma rede de máquinas flexíveis interligadas por um sistema automático de manuseio e armazenamento de materiais e controlados por um sistema distribuído de computação. Para ser considerado flexível o sistema de manufatura deve satisfazer a diversos critérios, sendo os essenciais a capacidade de processar diferentes tarefas e aceitar imediatamente mudanças no sistema de produção, sejam alterações no *mix* de peças produzidas ou no volume das mesmas

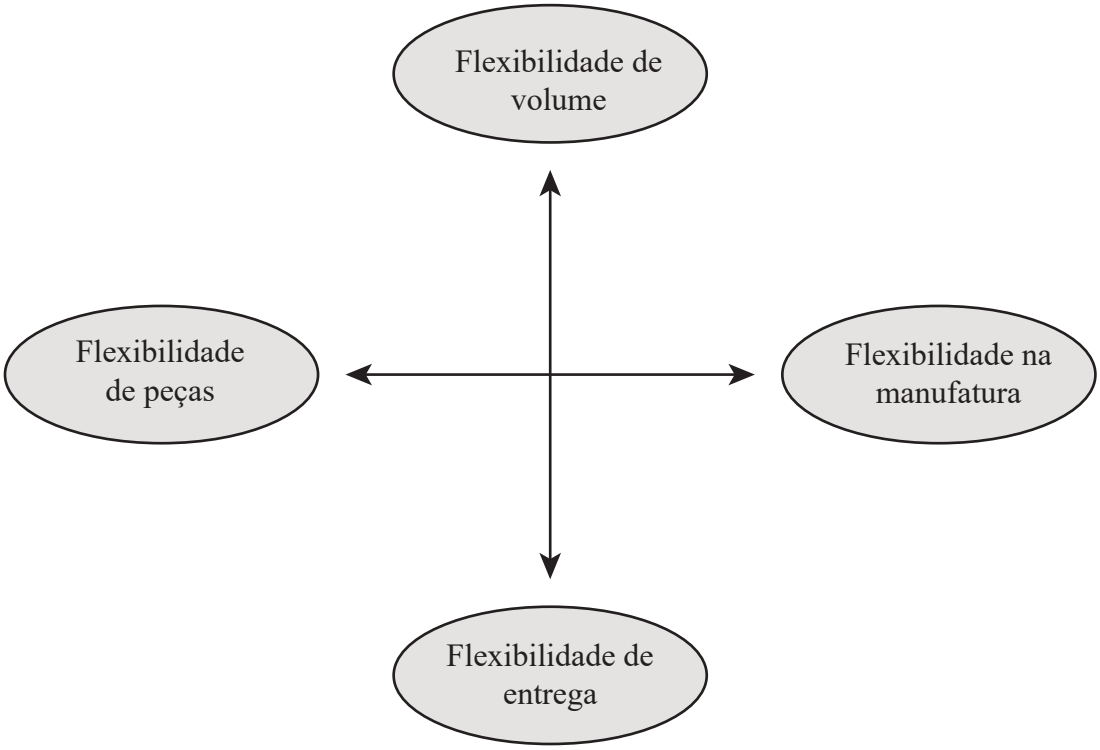


Figura 1 – Tipos de flexibilidade. Adaptado de Shivanand (2006).

(MIKELL, 2011). A satisfação destes dois critérios está diretamente relacionada ao uso de máquinas flexíveis. Uma máquina flexível é capaz de executar um grande número de tarefas diferentes como fresar, furar, lixar, entre outras. Desta maneira, uma mesma máquina que utilizando uma lâmina de corte, corta placas de alumínio, pode ser utilizada para perfurar trocando-se a lâmina de corte por uma broca de perfuração.

Cada tarefa processada por uma máquina flexível requer um conjunto específico de ferramentas. As ferramentas disponíveis para utilização pela máquina flexível durante o processamento de uma tarefa ficam alocadas no *magazine*, um compartimento específico com capacidade limitada. Em geral, a capacidade do *magazine* é o suficiente para armazenar todas as ferramentas necessárias para o processamento de uma tarefa isolada. Entretanto, o conjunto total de ferramentas pode ultrapassar esta capacidade. Assim, dado um conjunto de diferentes tarefas a serem processadas sequencialmente, pode ser necessário efetuar trocas de ferramentas no *magazine* antes que uma tarefa diferente possa ser processada.

SMFs estão presentes em diversos segmentos industriais, tais como, indústrias siderúrgicas, automotivas (MEERAN; MORSHED, 2012), farmacêuticas, químicas, de impressão de rótulos (GONZÁLEZ et al., 2015) e fabricação de calçados (ARANTES et al., 2004) entre outras (SHIRAZI; FRIZELLE, 2001). Já na década de 1980, estudos apontavam para um grande crescimento dos SMFs. Stecke (1983), percebendo que o gerenciamento de produção de um SMF era mais complexo que o de sistema de produção em linha, e que o problema de configuração geral do SMF era intratável, enumerou cinco problemas de

planejamento de produção que, resolvidos, possibilitariam maximizar a produção e reduzir custos.

Em síntese, o problema da *seleção de tarefas* consiste em, dado um conjunto de tarefas a serem processadas, determinar os subgrupos que devem ser processados imediatamente e simultaneamente considerando-se os requisitos de produção. O problema de *agrupamento de máquinas* consiste em agrupar as máquinas de forma que cada máquina em um grupo particular seja capaz de processar o mesmo conjunto de tarefas. O problema da *proporção de produção* consiste em determinar a proporção em que cada tipo de tarefa selecionada deve ser processada. O problema da *alocação de recursos* consiste em alocar os recursos necessários para o acabamento da produção e despacho dos bens produzidos. O último problema, *carregamento de tarefas e ferramentas* consiste em alocar as tarefas e as ferramentas necessárias entre os grupos de máquinas, respeitando-se as limitações das mesmas.

A partir desta classificação inicial, diversos problemas oriundos de SMFs têm sido estudado na literatura. Em especial, neste trabalho, serão estudados problemas de natureza combinatória derivados do problema de carregamento de tarefas e ferramentas, agora estudado sob o nome de sequenciamento de tarefas em máquinas ou *machine scheduling problem*. Trata-se de uma classe de problemas práticos que, em sua maioria, pertencem à classe $\mathcal{NP}$-Difícil (PINEDO, 2008), ou seja, problemas para os quais não se conhece algoritmo determinístico polinomial para suas soluções.

Conforme apresentado por Graham et al. (1979), os diferentes problemas de sequenciamento de tarefas em máquinas podem ser descritos pela notação $\alpha|\beta|\gamma$. O campo α descreve o ambiente das máquinas e contém apenas uma entrada. O campo β provê detalhes sobre as características e restrições do processamento e pode conter nenhuma, uma ou várias entradas. O campo γ descreve o objetivo a ser otimizado e geralmente contém apenas uma entrada.

Devido a ampla aplicação dos SMFs em indústrias de vários segmentos e tamanhos, diversos ambientes de máquinas são utilizados. Entre todos os ambientes possíveis, o mais simples é o ambiente contendo apenas uma única máquina flexível (representado pelo valor 1 no campo α) que processa todas as tarefas (PINEDO, 2008). A Figura 2, retirada de Mikell (2011), ilustra este ambiente. A máquina flexível é referida como "Centro de usinagem CNC" e é possível visualizar seu *magazine*, referido como "Armazenamento de ferramentas", além do sistema de manuseio e transporte de materiais do SMF.

Outro cenário de máquinas recorrente é o ambiente constituído por máquinas idênticas paralelas (representado pelo valor P_m no campo α). Neste ambiente, existem m máquinas idênticas operando em paralelo. Uma tarefa pode ser processada em qualquer uma das m máquinas com o mesmo tempo de processamento (PINEDO, 2008). O tempo de processamento de uma tarefa é conhecido a priori e corresponde ao intervalo de tempo que

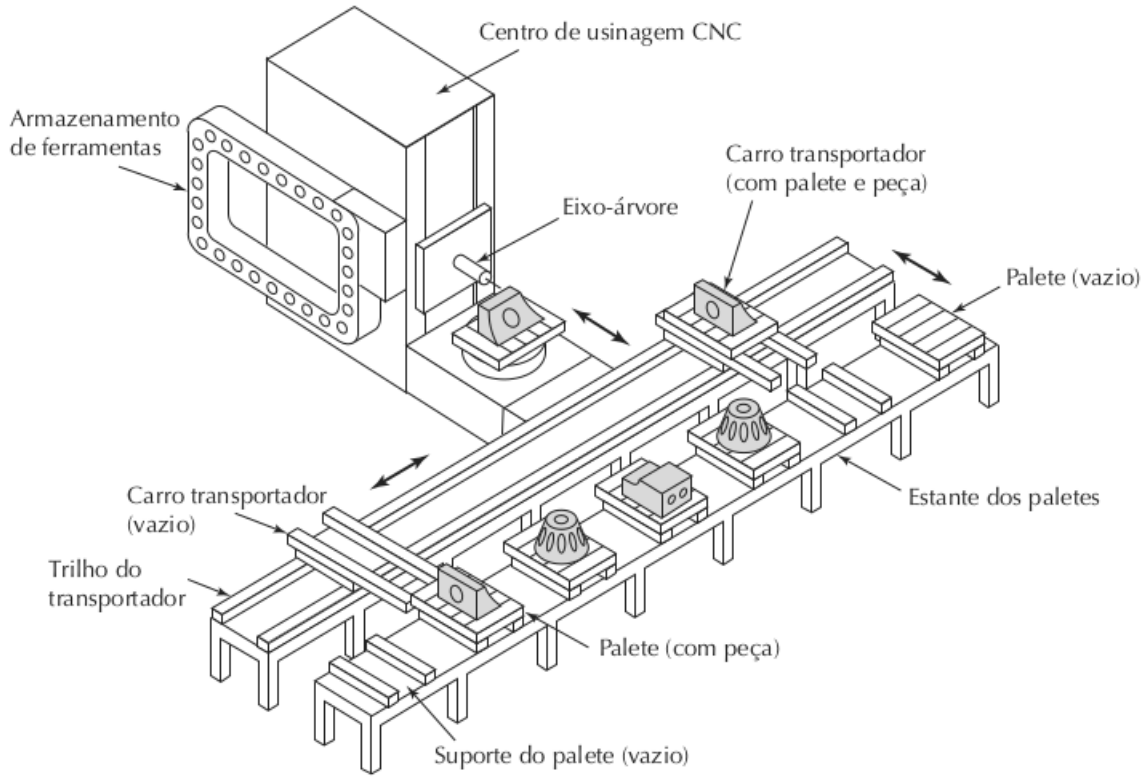


Figura 2 – Ambiente de manufatura flexível contendo apenas uma máquina flexível. Retirado de Mikell (2011).

uma tarefa permanece em uma máquina para o seu processamento. A Figura 3, retirada de Mikell (2011), ilustra o cenário de máquinas idênticas paralelas em um ambiente de manufatura flexível contendo três máquinas flexíveis idênticas e um sistema de manuseio e armazenagem de materiais.

Além dos ambientes industriais mencionados, outras possíveis configurações de ambiente para o campo α incluem máquinas flexíveis paralelas não-relacionadas (R_m) onde o tempo de processamento de cada tarefa depende da máquina onde ocorre o processamento, *flow shop* (F_m) onde as máquinas são dispostas em série e há uma rota pré-estabelecida passando por todas as máquinas que deve ser obedecida por cada tarefa a ser processada, entre outros (PINEDO, 2008).

De acordo com a especificidade de cada contexto, algumas restrições e características de processamento são definidas. Estas características são especificadas no campo β da classificação dada por Graham et al. (1979). Como exemplo de tais características podemos citar o tempo de disponibilização (r_j) que determina que j-ésima tarefa só pode ser processada após decorrido r_j unidades de tempo, preempção (*prmp*) que implica a não necessidade do processamento de uma tarefa ser ininterrupto, limitações de precedência (*prec*) que determina a existência de uma ordem de processamento específica a ser seguida, limitações de recursos (*res*) que indica a possibilidade de recursos necessários ao proces-

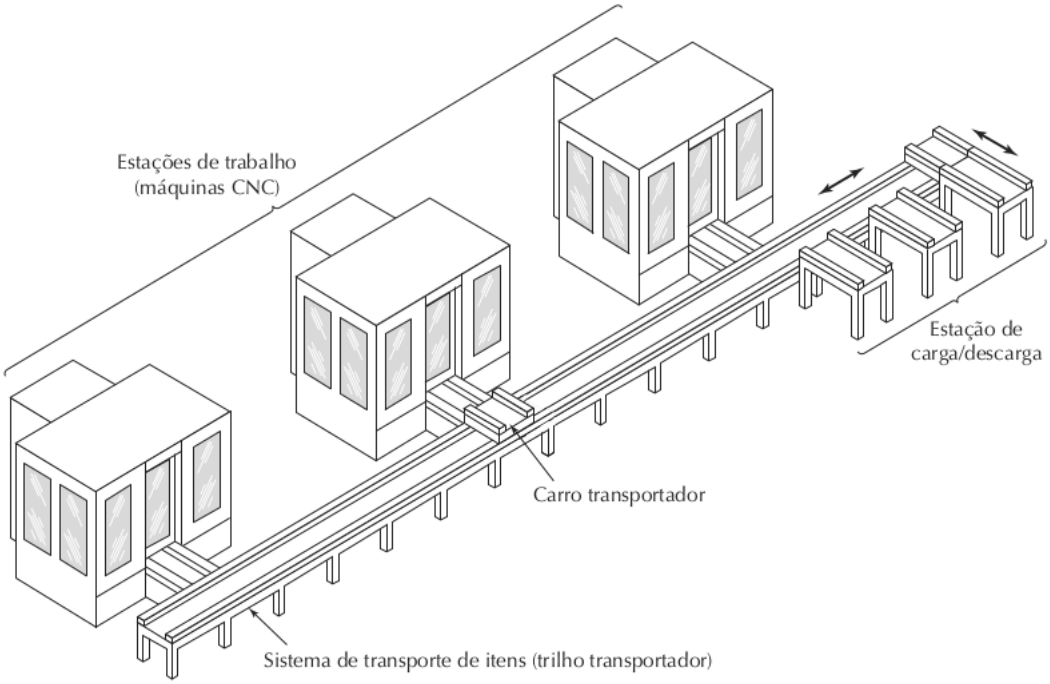


Figura 3 – Ambiente de manufatura flexível contendo três máquinas flexíveis idênticas operando em paralelo. Retirado de Mikell (2011).

samento de uma tarefa estarem indisponíveis em algum momento e tempos de preparo dependente da sequência (s_{jk}) que indica a possibilidade de serem necessárias operações de preparo da máquina entre o processamento subsequente das j-ésima e k-ésima tarefas.

Algumas funções objetivo comuns, especificadas pelo campo γ incluem *makespan* (C_{max}), atraso máximo (L_{max}) para os casos em que cada tarefa possui uma data de vencimento, ou seja, uma data de entrega prometida ao cliente, e tempo total de conclusão ponderado ($\sum w_j C_j$) onde o objetivo é minimizar a soma dos tempos de conclusão total (C_j) das n tarefas considerando-se um fator de prioridade (w_j) para cada tarefa j.

Problemas relacionados ao carregamento de tarefas e ferramentas dão origem a variados problemas combinatórios, de acordo com detalhes e especificidades de contexto de aplicação, como compartilhamento de recursos entre máquinas, uniformidade de ferramentas, data de disponibilização das tarefas, vida útil de ferramentas entre outros. O objetivo deste trabalho de pesquisa é abordar a otimização de processos operacionais em SMF, avançando o estado da arte em relação à solução de problemas diretamente aplicáveis neste contexto. Para este fim, problemas específicos e subproblemas relacionados são abordados nos capítulos posteriores deste trabalho. Além de uma formalização detalhada sobre suas especifidades, é apresentada uma ampla revisão bibliográfica, abrangendo desde os trabalhos seminais até os mais recentes trabalhos publicados. Ademais, são expostos os detalhes metodológicos das abordagens propostas, os experimentos computacionais realizados e as conclusões obtidas.

1.1 Justificativa

Problemas derivados do sequenciamento de tarefas possuem grande importância teórica pois pertencem a classe de problemas $\mathcal{NP}$-Difíceis (PINEDO, 2008), ou seja, problemas para os quais não existem métodos computacionais eficientes, exceto se $P = NP$ (GAREY; JOHNSON, 1979a). A relevância prática destes problemas é demonstrada por sua ampla aplicação em diversos segmentos. Um exemplo de aplicação ocorre em indústrias de impressão de rótulos em embalagens, com o objetivo de minimizar o número de interrupções necessárias para trocar os cartuchos de tinta, entre as trocas de cores (BURGER et al., 2015). Um segundo exemplo ocorre em indústrias eletrônicas de montagem de microchips em placas de circuitos impressos, visando a minimização do número de trocas de alimentadores de componentes nas máquinas montadoras (HOP; NAGARUR, 2004). Outras aplicações incluem desde a definição de rotas para o processamento de requisições entre servidores móveis (PRIVAULT; FINKE, 2000), até o sequenciamento de produção na fabricação de para-brisas de automóveis (MEERAN; MORSHED, 2012), entre outros.

O estudo e abordagem adequada de tal conjunto de problemas tem o potencial para maximizar a produção e reduzir os custos operacionais em SMFs (STECKE, 1983). Por exemplo, problemas relacionados a minimização de trocas de ferramentas possibilitam reduzir o tempo ocioso das máquinas flexíveis. Para que estas trocas de ferramentas possam ocorrer, é necessário parar a máquina temporariamente. As ferramentas que serão inseridas precisam ser movidas do depósito até a máquina. Havendo a necessidade de liberar espaço no *magazine*, deve-se selecionar quais ferramentas serão removidas, removê-las, inserir as novas ferramentas e então mover as ferramentas retiradas de volta ao depósito. De fato, operações advindas das trocas de ferramentas consomem mais tempo do que qualquer outra operação de *setup* em um SMF (MELNYK; GHOSH; RAGATZ, 1989; AGAPIOU, 1991; ZHANG; HINDUJA, 1995; HOP; NAGARUR, 2004).

Embora sistemas de manufatura reconfiguráveis (*reconfigurable manufacturing systems*, RMS) sejam uma tendência emergente nas fábricas inteligentes, pois permitem a rápida integração de novas tecnologias e funcionalidades (BI et al., 2008; BRETTEL et al., 2014), SMFs continuam sendo uma área promissora para futuros sistemas de produção (MEHRABI et al., 2002). Em ambos os contextos, RMS e SMF, a minimização do tempo de configuração necessário ao sistema de produção é crucial (CALMELS, 2019).

Problemas diretamente aplicáveis ao setor industrial, como os abordados neste trabalho, são especialmente importantes para a economia nacional. Segundo relatório publicado pela Confederação Nacional da Indústria (2020), mesmo diante de um cenário de retração na última década, o setor industrial foi responsável por 20,9% do produto interno bruto (PIB) brasileiro em 2019, com destaque especial para a indústria de transformação, sendo esta responsável por 11,0% do PIB.

Entre os estados brasileiros, Minas Gerais destaca-se ocupando a segunda posição como maior contribuidor do PIB industrial nacional. De acordo com o mesmo relatório, com dados referentes ao ano de 2017, Minas Gerais é responsável por 10,7% do PIB industrial brasileiro. Em Minas Gerais, setores industriais com potencial direto para adoção de SMFs, como metalurgia, e relacionados à produção de veículos automotores, produtos químicos, produtos de metal, celulose, calçados, eletrônicos, farmacêuticos e impressão, somados, são responsáveis por 24,2% do PIB industrial do estado (Confederação Nacional da Indústria, 2017).

Por fim, conforme reportado por Calmels (2019), o número de publicações em temas correlatos ao sequenciamento de tarefas em SMFs têm crescido consistentemente desde 1988, demonstrando a relevância e atualidade do tema.

1.2 Objetivos

Este trabalho de pesquisa propõe o estudo e abordagem computacional de diferentes problemas combinatórios derivados do problema de sequenciamento de tarefas em máquinas. Particularmente, abordam-se os problemas desta natureza oriundos de SMFs. Diferentes abordagens são propostas em acordo com as especificidades de cada problema. São objetivos específicos:

- Realizar pesquisa para geração de embasamento teórico sobre SMFs e os diversos problemas abordados;
- Gerar uma revisão bibliográfica atual e abrangente para cada problema tratado;
- Propor métodos computacionais consistentes e competitivos para a resolução dos problemas tratados;
- Analisar criticamente os resultados obtidos considerando-se as instâncias disponibilizadas na literatura e compará-los com o estado da arte.

1.3 Organização do texto

O texto desta tese encontra-se organizado como descrito a seguir. O Capítulo 2 apresenta em detalhes dois problemas básicos em SMFs: o problema de minimização de trocas de ferramentas e o sequenciamento de tarefas em máquinas idênticas paralelas com restrições de ferramentas. O referencial teórico destes problemas serve como base para os demais problemas abordados. Os capítulos 3 e 4 apresentam abordagens a problemas relacionados ao sequenciamento de tarefas em máquinas flexíveis. Para cada problema

são apresentados uma descrição detalhada, revisão bibliográfica abrangendo as principais extensões e aplicações, metodologia proposta, experimentos computacionais e conclusões.

O Capítulo 3 aborda o problema de minimização de blocos de uns. Este capítulo é fortemente baseado no artigo Soares et al. (2020). O Capítulo 4 aborda o problema de sequenciamento de tarefas em máquinas paralelas com limitação de recursos. O Capítulo 5 apresenta as conclusões gerais e os trabalhos futuros. Por fim, no Apêndice A são listados os artigos publicados durante o programa de doutorado.

2 Problemas fundamentais de carregamento de tarefas e ferramentas

Dada a ampla utilização de SMFs em diferentes segmentos industriais, diversos problemas com origem neste sistema têm sido estudados ao longo dos anos. Por mais específicos que sejam, estes problemas possuem sua origem relacionada à problemas que ocorrem em ambientes de manufatura flexível contendo uma ou várias máquinas. Assim, neste capítulo são descritos dois problemas, um em cada ambiente, que servirão de base teórica para os demais problemas abordados.

2.1 Minimização de Trocas de Ferramentas

Conforme mencionado no Capítulo 1, considerando-se apenas SMFs, o ambiente industrial mais simples possível é o constituído por apenas uma máquina flexível. Neste ambiente ocorre o problema de Minimização de Trocas de Ferramentas (ou *job sequencing and tool switching problem*, SSP), cujo objetivo é determinar uma sequência para o processamento de n tarefas que minimize o número de trocas de ferramentas necessárias. Este problema é constituído pela combinação de dois outros problemas, o sequenciamento do processamento das tarefas na máquina e a determinação do plano de trocas de ferramentas. O primeiro problema é $\mathcal{NP}$-Difícil (CRAMA et al., 1994), sendo responsável pela complexidade do SSP. Após a definição da sequência de processamento das tarefas, o plano de trocas de ferramentas pode ser determinado em tempo determinístico polinomial utilizando-se a política de manter no *magazine* as ferramentas que serão utilizadas mais cedo, denominada *Keep Tool Needed Soonest* (KTNS) e introduzida por Tang e Denardo (1988).

Embora as bases teóricas do SSP tenham sido formalmente definidas pelos trabalhos de Tang e Denardo (1988) e Bard (1988), ambos trabalhos baseiam-se na publicação seminal de Tang (1986). Formalmente, dado um SMF contendo uma máquina flexível com capacidade para armazenar até C ferramentas simultaneamente em seu *magazine*, um conjunto $J = \{1, \ldots, n\}$ de tarefas a serem processadas, um conjunto $T = \{1, \ldots, r\}$ de ferramentas e um subconjunto T_j ($T_j \in T$) de ferramentas necessárias para processar a tarefa j ($j \in J$), o SSP consiste em determinar o sequenciamento do processamento das tarefas na máquina flexível de forma que este resulte no menor número possível de trocas de ferramentas.

De acordo com a classificação dada por Graham et al. (1979), o SSP é caracterizado pela classe $1|s_{jk}|C_{max}$. Especificamente, o valor s_{jk} definido para o campo β indica um

tempo de preparo dependente de toda a sequência de tarefas e não apenas das j-ésima e k-ésima tarefas. Visto que a capacidade do *magazine* pode ser superior a quantidade de ferramentas necessárias para o processamento de uma tarefa isolada, é possível que ferramentas não utilizadas pela tarefa atual, mas que serão necessárias em tarefas subsequentes, sejam previamente carregadas. Esta característica difere o SSP de outros pertencentes à mesma classe.

Em seu contexto considerado padrão, o SSP possui as seguintes características (BARD, 1988; CRAMA et al., 1994): (i) todas as ferramentas são consideradas idênticas, portanto, podem ocupar qualquer posição no *magazine*; (ii) somente uma ferramenta pode ser trocada por vez e o tempo de troca é constante e idêntico para todas as ferramentas; (iii) o conjunto de tarefas e o subconjunto de ferramentas necessárias para o processamento de cada tarefa é conhecido *a priori*; (iv) o número de ferramentas necessárias para processar uma tarefa é menor ou igual a capacidade do *magazine* da máquina; e, (v) não são consideradas trocas de ferramentas originadas por quebras ou desgastes das ferramentas.

Para que uma determinada tarefa j possa ser processada, é necessário que o subconjunto de ferramentas T_j ($|T_j| \leq C$) seja previamente carregado no *magazine* da máquina. Uma instância válida para o SSP consiste da lista de tarefas a serem processadas, do conjunto de ferramentas necessárias para o processamento de cada tarefa e da capacidade do *magazine* da máquina. A Tabela 1 apresenta um modelo de instância válida em que se considera $C = 6$, $n = 6$ e $T = \{1, \ldots, 10\}$.

Tabela 1 – Exemplo de instância do SSP.

Tarefas	Ferramentas
1	5, 6 ,7, 9, 10
2	2, 4, 6
3	1, 2, 3, 5, 6
4	1, 3, 5, 7
5	3, 6, 7, 8, 9
6	2, 8

É possível utilizar uma matriz binária Q para representar uma instância do SSP. Nesse caso, a matriz resultante é denominada *matriz de ferramentas*. Cada linha da matriz corresponde a uma tarefa e cada coluna corresponde a uma ferramenta. Se uma tarefa j requer a ferramenta t, então $q_{jt} = 1$. Caso contrário, $q_{jt} = 0$. A matriz de ferramentas Q, para o exemplo de instância mostrado na Tabela 1, pode ser vista na Tabela 2.

A solução do problema é obtida a partir da permutação π das colunas do *plano de trocas de ferramentas* que considera a produção dividida em estágios. A cada estágio, representado pelas colunas do plano, uma tarefa é processada. As linhas do plano de ferramentas indicam as ferramentas carregadas no *magazine* antes do processamento da tarefa alocada para o estágio atual. O plano de troca de ferramentas de uma máquina

Tabela 2 – Exemplo de matriz de ferramentas.

Tarefas	Ferramentas									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	1	1	1	0	1	1
2	0	1	0	1	0	1	0	0	0	0
3	1	1	1	0	1	1	0	0	0	0
4	1	0	1	0	1	0	1	0	0	0
5	0	0	1	0	0	1	1	1	1	0
6	0	1	0	0	0	0	0	1	0	0

flexível pode ser representado por uma matriz binária $R^{\pi} = \{r_{tj}\}$, com $t \in \{1, \ldots, r\}, j \in \{1, \ldots, n\}$, cujas colunas obedecem a ordem estabelecida por π. Um elemento $r_{tj} = 1$ indica que a ferramenta t está carregada no *magazine* da máquina flexível durante o processamento da tarefa j, e $r_{tj} = 0$ o caso contrário.

Como forma de ilustração, considere-se a permutação $\pi = \{1, 3, 5, 2, 4, 6\}$. A Tabela 3 apresenta o plano de trocas de referente a esta permutação. As ferramentas removidas ou inseridas imediatamente antes do início do processamento de uma tarefa estão sublinhadas, por exemplo, imediatamente antes que a tarefa 2 fosse processada, no quarto estágio, a ferramenta 8 foi removida e a ferramenta 4 inserida, portanto, ambas ferramentas aparecem sublinhadas.

Tabela 3 – Exemplo do plano de troca de ferramentas.

Ferramentas	Tarefas					
	1	3	5	2	4	6
1	0	$\underline{1}$	$\underline{0}$	0	$\underline{1}$	$\underline{0}$
2	0	$\underline{1}$	1	1	1	1
3	$\underline{1}$	0	1	1	1	1
4	0	0	0	$\underline{1}$	$\underline{0}$	0
5	$\underline{1}$	1	$\underline{0}$	0	$\underline{1}$	1
6	$\underline{1}$	1	1	1	1	1
7	$\underline{1}$	$\underline{0}$	$\underline{1}$	1	1	1
8	0	0	$\underline{1}$	$\underline{0}$	0	$\underline{1}$
9	$\underline{1}$	1	1	1	$\underline{0}$	0
10	$\underline{1}$	$\underline{0}$	0	0	0	0

Conforme demonstrado pela Tabela 3, dada a permutação $\pi = \{1, 3, 5, 2, 4, 6\}$, a primeira tarefa a ser processada, $j = 1$, necessita das ferramentas $\{5, 6, 7, 9, 10\}$. Como a quantidade de ferramentas em T_1 (5) é inferior a capacidade total do *magazine* ($C = 6$), uma ferramenta adicional, não necessária para o processamento da tarefa atual, mas necessária em processamentos futuros, será carregada. Por exemplo, a ferramenta 3. Assim, o *magazine* da máquina será carregado inicialmente com as ferramentas $\{3, 5, 6, 7, 9, 10\}$. As inserções de ferramentas iniciais na máquina, requerem *setup* e por isso são consideradas trocas. Ao final do carregamento inicial, temos 6 trocas de ferramentas.

Após o processamento da primeira tarefa, antes que a tarefa 3 seja processada é necessário inserir no *magazine* as ferramentas $\{1, 2\} \in T_3$. Como não há espaço no *magazine*, duas ferramentas precisam ser removidas. Existem três ferramentas aptas a serem retiradas do *magazine*, $\{7, 9, 10\} \notin T_3$. Utilizando-se a política KTNS a ferramenta 10 é seleciona pois não será utilizada por nenhuma outra tarefa. As ferramentas 7 e 9 serão necessárias ao mesmo tempo, para o processamento da tarefa 5. De acordo com a política utilizada, empates devem ser decididos aleatoriamente. Assim, seleciona-se a ferramenta 7. As ferramentas 7 e 10 são removidas e as ferramentas 1 e 2 inseridas no *magazine*, gerando duas trocas adicionais e 8 no total.

Para que a tarefa 5 possa ser processada no estágio seguinte, as ferramentas $\{7, 8\} \in T_5$ precisam ser inseridas no *magazine*. Como candidatas a serem removidas temos as ferramentas $\{1, 2, 5\} \notin T_5$. A ferramenta 2 será necessária ao processamento da tarefa 2 e por isso será mantida no *magazine*. As ferramentas 1 e 5 são substituídas pelas ferramentas 7 e 8, gerando duas trocas adicionais e totalizando 10 trocas de ferramentas.

Antes que a tarefa 2 possa ser processada, a ferramenta $4 \in T_2$ precisa ser inserida no *magazine*. Como candidatas a serem removidas temos as ferramentas $\{3, 7, 8, 9\} \notin T_2$. As ferramentas $\{8, 9\}$ não serão utilizadas por nenhuma das tarefas a serem processadas nos próximos estágios e por isso possuem prioridade de remoção. A ferramenta 8 é aleatoriamente selecionada e substituída pela ferramenta 4, gerando uma troca adicional e totalizando 11 trocas de ferramentas.

Para o processamento sequencial da tarefa 4, as ferramentas $\{1, 5\} \in T_4$ precisam ser inseridas no *magazine*. As ferramentas $\{2, 4, 6, 9\} \notin T_4$ são candidatas a serem removidas. Apenas a ferramenta 2 será necessária no processamento de tarefas subsequentes, portanto, as ferramentas 4 e 9 são selecionadas aleatoriamente e substituídas pelas ferramentas 1 e 5, gerando duas trocas adicionais e totalizando 13 trocas de ferramentas.

Para o processamento da tarefa 6, será necessário inserir no magazine a ferramenta $8 \in T_6$. Como trata-se da última tarefa, qualquer ferramenta não pertencente a T_6 pode ser removida e substituída pela ferramenta necessária. Aleatoriamente, a ferramenta 1 é selecionada e substituída pela ferramenta 8, gerando uma troca adicional. No total, para processar todas as tarefas na ordem definida pela permutação π, são necessárias 14 trocas de ferramentas.

Uma solução representada pela matriz R^π é avaliada pela Equação (2.1) proposta por Crama et al. (1994). Esta função calcula o número de inversões de 0 para 1, que representam a inserção de ferramentas na máquina. Para que as inserções iniciais sejam consideradas, a matriz R^π possui uma coluna inicial fictícia com todos os elementos iguais

a zero, ou seja, $R^{\pi}_{t0} = 0$.

$$Z_{SSP}(R^{\pi}) = \sum_{t \in T} \sum_{j \in J} r^{\pi}_{tj}(1 - r^{\pi}_{tj-1}) \tag{2.1}$$

Conforme mencionado, o objetivo do SSP é obter uma permutação $\pi \in \Pi$ para o processamento de tarefas que minimize o número total de trocas de ferramentas, em que Π é o conjunto de todas as permutações possíveis. A função objetivo para o SSP é representada pela Equação (2.2).

$$\min_{\pi \in \Pi} \left\{ Z^{\pi}_{SSP}(R) \right\} \tag{2.2}$$

Com o objetivo de minimizar o número de trocas de ferramentas, deve-se evitar que ferramentas que serão necessárias em etapas posteriores do plano de produção sejam removidas do *magazine*. Tais remoções evidenciam-se pela presença de *descontinuidades* no plano de trocas de ferramentas. Uma descontinuidade ocorre sempre que um conjunto de elementos nulos aparece entre dois conjuntos de elementos não nulos em uma mesma linha da matriz binária. O estudo de tal particularidade, com o objetivo de minimizar o número de descontinuidades, caracteriza o problema de minimização de blocos de uns consecutivos (ou *Consecutive Block Minimization*, CBM), que aparece como subproblema do SSP. Portanto, o progresso no estudo de tal problema possui potencial para evoluir o estudo de problemas derivados do carregamento de tarefas e ferramentas. Devido a sua importância, o CBM é abordado no Capítulo 3.

2.1.1 Trabalhos relacionados

A literatura específica sobre o SSP remonta à década de 1980. Desde então, diversos autores têm abordado o problema e suas numerosas variações (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999; ALLAHVERDI et al., 2008; ALLAHVERDI, 2015; CALMELS, 2019). A seguir, apresenta-se em ordem cronológica as principais contribuições para o tema.

O SSP foi formalmente introduzido e definido por Tang e Denardo (1988). Neste trabalho o problema é modelado e resolvido como uma instância do problema do caixeiro viajante (ou *Traveling Salesman Problem*, TSP). Além disso, os autores apresentam uma política para determinar, em tempo determinístico polinomial, o plano ótimo de troca de ferramentas para uma dada sequência fixa de tarefas. Esta política consiste em manter no *magazine* da máquina as ferramentas que serão novamente necessárias mais cedo dentro do plano de produção, denominada *Keep Tool Needed Soonest*, KTNS.

Bard (1988) modelou o SSP como um problema de programação linear não inteira e apresentou um método de *branch-and-bound* para resolver uma relaxação lagrangiana proposta para o problema. Os experimentos computacionais utilizaram um conjunto de instâncias proposto no próprio trabalho. Visando um baixo tempo computacional, o número de iterações do método proposto foi limitado, assim, foram reportadas soluções que representam apenas ótimos locais para o problema.

Crama et al. (1994) provam que o SSP pertence à classe de problemas $\mathcal{NP}$-Difícil para qualquer instância contendo *magazine* com capacidade superior ou igual a dois. Neste trabalho, novamente o SSP foi modelado como uma instância do TSP. Tal modelagem tornou-se recorrente na literatura (HERTZ et al., 1998; DJELLAB; DJELLAB; GOURGAND, 2000; SHIRAZI; FRIZELLE, 2001; LAPORTE; SALAZAR-GONZALEZ; SEMET, 2004; AHMADI et al., 2018). Entretanto, em trabalhos teóricos, Yanasse e Lamosa (2006a), Yanasse e Lamosa (2006b) reportam que tal abordagem é ineficiente, pois o SSP e o TSP não são equivalentes, propondo a modelagem do problema como uma instância do problema do caixeiro viajante generalizado (ou *Generalized Traveling Salesman Problem*, GTSP). Além disso, o trabalho apresentado por Azevedo e Carvalho (2017) demonstram na prática que as métricas existentes na literatura para definição das matrizes de distâncias utilizadas na modelagem do SSP como TSP, sumarizados em Hertz et al. (1998), produzem soluções de baixa qualidade, sendo necessário o desenvolvimento de novas métricas para que tal modelagem possa ser utilizada de forma eficiente.

Djellab, Djellab e Gourgand (2000) modelam o SSP utilizando um hipergrafo. Para solução da modelagem é apresentado uma implementação da heurística gulosa de inserção da hiperaresta de menor custo. Os experimentos computacionais utilizaram um conjunto de instâncias proposto no próprio artigo e os resultados não foram comparados com outros métodos.

Shirazi e Frizelle (2001) apresentam um estudo comparando o escalonamento manual utilizado por indústrias da época com o resultado heurístico obtido a partir da modelagem do SSP como uma instância do TSP. Foram utilizadas seis métodos heurísticos gulosos recorrentes na abordagem ao TSP, a saber: *simple greedy*; *multiple start greedy*; *best position insertion*; *shortest edge*, e duas variações da heurística *farthest insertion*. Os experimentos computacionais utilizaram instâncias reais e os resultados reportados demonstraram que a abordagem heurística possibilitava melhorias significativas.

Seis implementações diferentes da metaheurística busca tabu foram propostas por Al-Fawzan e Al-Sultan (2003) para o SSP. As implementações diferem-se uma das outras pelas estratégias utilizadas na etapa de intensificação da busca. Os experimentos computacionais utilizaram instâncias aleatoriamente geradas no próprio artigo e os resultados foram comparados apenas entre si.

Um novo modelo de programação linear inteira para o SSP foi apresentado por

Laporte, Salazar-Gonzalez e Semet (2004). O modelo apresentado domina o modelo proposto por Tang e Denardo (1988). Além disso, são apresentados e comparados dois algoritmos para o problema, um método de *branch-and-bound* e um de *branch-and-cut*. Os experimentos realizados utilizaram instâncias geradas no próprio artigo. O método *branch-and-cut* mostrou-se apto a resolver de forma ótima instâncias com até nove tarefas e o método *branch-and-bound* instâncias com até 25 tarefas.

Zhou, Xi e Cao (2005) abordam o SSP com um algoritmo de *beam search*. Os experimentos computacionais utilizaram 80 instâncias geradas aleatoriamente conforme os parâmetros definidos por Bard (1988). Os resultados obtidos pelo método proposto superam os reportados pelo método de *branch-and-bound* apresentado por Bard (1988).

Diferentes implementações de algoritmos meméticos foram propostas para o SSP por Amaya, Cotta e Fernández (2008), Amaya, Cotta e Fernández-Leiva (2011), Amaya, Cotta e Fernandez-Leiva (2012), Amaya, Cotta e Fernández-Leiva (2013). Sempre utilizando instâncias geradas aleatoriamente de acordo com os parâmetros estabelecidos na literatura prévia, estes trabalhos limitam-se a comprar os resultados obtidos entre as diferentes implementações propostas em um mesmo trabalho e suas versões anteriores, não sendo possível aferir a qualidade dos métodos apresentados frente às demais abordagens presentes na literatura.

Yanasse, Rodrigues e Senne (2009) apresentam um algoritmo enumerativo para solucionar o SSP. Os experimentos computacionais realizados demonstram que o método proposto supera os resultados reportados por Laporte, Salazar-Gonzalez e Semet (2004). Em um trabalho posterior, Senne e Yanasse (2009) apresentam três implementações de *beam search* baseadas no mesmo esquema enumerativo. Os experimentos computacionais utilizaram instâncias geradas aleatoriamente no próprio trabalho, seguindo o padrão proposto por Laporte, Salazar-Gonzalez e Semet (2004). Os resultados obtidos pelas implementações propostas não foram comparados com outros métodos.

Chaves, Yanasse e Senne (2012) modelam o SSP como um grafo no qual os vértices representam as ferramentas e os arcos as tarefas. Para cada tarefa existem arcos rotulados interligando as ferramentas necessárias para seu processamento. Uma heurística construtiva é utilizada para a geração de uma solução viável a partir desta representação e a metaheurística *iterated local search* (ILS) é aplicada sobre esta solução em uma etapa de refinamento. Os experimentos computacionais utilizaram cinco conjuntos de instâncias gerados aleatoriamente por Yanasse, Rodrigues e Senne (2009). Os resultados obtidos pela ILS são utilizados como limitantes superiores para o método enumerativo proposto por Yanasse, Rodrigues e Senne (2009), resultando em reduções significativas no número de nós gerados na árvore de busca e no tempo computacional.

Baseado no modelo introduzido por Laporte, Salazar-Gonzalez e Semet (2004), Catanzaro, Gouveia e Labbé (2015) apresentam dois novos modelos de programação linear

inteira para o SSP. Os experimentos computacionais utilizaram as instâncias geradas por Laporte, Salazar-Gonzalez e Semet (2004) e os resultados reportados demonstraram que as novas restrições e relaxações propostas pelos autores resultam em melhoria das soluções quando comparadas as reportadas pelos modelos anteriores da literatura.

Chaves et al. (2016) apresentam um método híbrido combinando um algoritmo de busca de *clusters* (ou Clustering Search, CS) com a metaheurística algoritmo genético de chaves aleatórias viciadas (ou *Biased random-key genetic algorithm*, BRKGA). Os experimentos computacionais utilizaram 1350 instâncias introduzidas por Yanasse, Rodrigues e Senne (2009) e 160 instâncias introduzidas por Crama et al. (1994). O método proposto foi capaz de superar ou igualar o melhor valor conhecido para todas as instâncias.

Paiva e Carvalho (2017) apresentam uma nova representação em grafo, uma heurística construtiva e uma nova implementação da ILS para o SSP. Entre as buscas locais utilizadas pela ILS, os autores introduzem uma busca local por agrupamento de blocos de uns consecutivos em uma matriz binária, explorando a conexão entre o SSP e o CBM. Além das 1510 instâncias de Yanasse, Rodrigues e Senne (2009) e Crama et al. (1994), os experimentos computacionais utilizaram ainda 160 instâncias propostas por Catanzaro, Gouveia e Labbé (2015). Os resultados obtidos para as instâncias de Yanasse, Rodrigues e Senne (2009) e Crama et al. (1994) foram comparados aos reportados por Chaves et al. (2016). Os resultados obtidos para as instâncias de Catanzaro, Gouveia e Labbé (2015) foram comparados aos melhores resultados conhecidos para as mesmas. De acordo com os experimentos realizados, o ILS reportou soluções melhores ou equivalentes para todas as instâncias consideradas.

Ahmadi et al. (2018) modelam o SSP como uma instância do problema do caixeiro viajante de segunda ordem (ou *Traveling Salesman Problem of Second Order*, 2-TSP) (JÄGER; MOLITOR, 2008) e o resolvem com uma implementação do algoritmo genético *q-learning*. Os experimentos computacionais utilizaram as instâncias de Catanzaro, Gouveia e Labbé (2015) e Crama et al. (1994). Em comparação ao trabalho de Paiva e Carvalho (2017), são reportadas melhorias marginais na qualidade das soluções. Entretanto, os autores não apresentam detalhes da métrica utilizada para determinação das matrizes de distância, impossibilitando uma melhor análise da mesma.

Silva, Chaves e Yanasse (2020) apresentam uma nova formulação linear inteira para o SSP utilizando um modelo de fluxo *multicommodity*. Os experimentos utilizaram as instâncias de Yanasse, Rodrigues e Senne (2009) e Catanzaro, Gouveia e Labbé (2015). Os resultados reportados superam os obtidos por modelos anteriores (TANG; DENARDO, 1988; LAPORTE; SALAZAR-GONZALEZ; SEMET, 2004; CATANZARO; GOUVEIA; LABBé, 2015) em número de soluções ótimas reportadas, qualidade do limite inferior e tempo de execução. Entretanto, considerando-se o tempo limite estabelecido em 3600 segundos, a modelagem não é capaz de resolver os maiores problemas dentre as instâncias

utilizadas.

Mecler, Subramanian e Vidal (2021) abordam o SSP utilizando um algoritmo de pesquisa genética híbrida (ou *Hybrid Genetic Search*, HGS). Além do *benchmark* de instâncias disponível na literatura (YANASSE; RODRIGUES; SENNE, 2009; CRAMA et al., 1994; CATANZARO; GOUVEIA; LABBé, 2015), os experimentos computacionais utilizaram um novo conjunto contendo 60 instâncias propostas no próprio artigo. Este novo conjunto é constituído exclusivamente por instâncias grandes, com o número de tarefas variando entre 50 e 70. Os resultados reportados foram comparados com os obtidos pelos métodos apresentados por Chaves et al. (2016), Paiva e Carvalho (2017) e Ahmadi et al. (2018). O método proposto reportou melhores resultados médios para todos os conjuntos de instâncias, tornando-se o atual estado da arte para o SSP.

2.2 Sequenciamento de tarefas em máquinas idênticas paralelas com restrições de ferramentas

O sequenciamento de tarefas em um ambiente composto por máquinas paralelas constitui o denominado Problema de Escalonamento em Máquinas Paralelas (*Scheduling Problem of Parallel Machines*, SPM), introduzido por McNaughton (1959), cujo objetivo é minimizar o maior tempo de processamento entre todas as máquinas do ambiente (ou *makespan*). O tempo de processamento de uma máquina é igual a soma dos tempos de processamento de todas as tarefas alocadas a ela. No caso específico dos SMFs, entre a execução sequencial de duas tarefas podem ser necessárias operações de configuração da máquina flexível, como trocas de ferramentas ou operações de limpeza. Estes casos implicam em tempo de preparo das máquinas (ou *setup time*), e este deve ser acrescentado ao tempo de processamento da máquina. Da junção do SPM com o SSP, surge o Problema de Escalonamento de Tarefas em Máquinas Paralelas Idênticas com Restrições de Ferramentas (ou *Identical Parallel Machines With Tooling Constraints*, IPMTC), abordado primeiramente por Stecke (1983).

Considere um SMF contendo um conjunto $M = \{1, \ldots, m\}$ de máquinas flexíveis idênticas paralelas, cada uma com capacidade para comportar até C ferramentas simultaneamente em seu *magazine*, um conjunto $J = \{1, \ldots, n\}$ de tarefas a serem processadas, um conjunto $T = \{1, \ldots, r\}$ de ferramentas e um subconjunto T_j ($T_j \in T$) de ferramentas necessárias para processar a tarefa j ($j \in J$), um tempo de processamento p_j ($j \in J$) para cada tarefa e um tempo constante $\bar{p}$ para troca de cada ferramenta. O IPMTC consiste em determinar a alocação e o sequenciamento das tarefas nas m máquinas de forma que este sequenciamento resulte no menor tempo de produção total, visando a minimização do *makespan*, representado por Δ. Especificamente para o IPMTC, o tempo processamento de uma máquina é igual a soma dos tempos de processamento de todas as tarefas alocadas

a ela, acrescido do número de trocas de ferramentas, multiplicado por $\bar{p}$.

O IPMTC pertence à classe de problemas $\mathcal{NP}$-Difícil (CRAMA et al., 1994) e foi formalmente definido por Fathi e Barnette (2002). De acordo com a classificação de Graham et al. (1979), o IPMTC pertence à classe $Pm|s_{jk}|C_{max}$. O campo β possui as mesmas características detalhadas para o SSP. Em sua formulação padrão, o IPMTC possui as seguintes características: (i) todas as tarefas são conhecidas *a priori*; (ii) não há ordem de precedência entre as tarefas; (iii) o processamento de uma tarefa não pode ser interrompido antes do seu final; (iv) cada máquina possui um conjunto completo de ferramentas; (v) a capacidade do *magazine* das máquinas é limitada, porém, grande o suficiente para processar qualquer tarefa individualmente; (v) o tempo necessário para trocar uma ferramenta é constante e idêntico para todas as ferramentas; (vi) as ferramentas podem ocupar qualquer posição disponível no magazine; e, (vii) o tempo necessário para a configuração inicial da máquina não é considerado.

Uma instância do IPMTC consiste em um conjunto de tarefas a serem processadas, seus respectivos tempos de processamento, o subconjunto de ferramentas necessário para cada tarefa, além do número de máquinas flexíveis, capacidade do *magazine* e o tempo necessário para a troca de cada ferramenta. A Tabela 4 apresenta um modelo de instância válida em que se considera $C = 6$, $m = 2$ e $\bar{p} = 1$.

Tabela 4 – Exemplo de instância do IPMTC.

Tarefas	Tempo de processamento	Ferramentas
1	10	5, 6 ,7, 9, 10
2	5	2, 4, 6
3	10	1, 2, 3, 5, 6
4	5	1, 3, 5, 7
5	5	3, 6, 7, 8, 9
6	10	2, 8

Conforme apontado por Fathi e Barnette (2002), a solução do IPMTC passa obrigatoriamente por três etapas: (i) alocação das tarefas às máquinas; (ii) o sequenciamento das tarefas em cada máquina; e, (iii) a determinação do plano de trocas de ferramentas para a sequência dada em cada máquina. Após o sequenciamento das tarefas em cada máquina, o problema se reduz a m instâncias do SSP. Então, o plano de trocas de ferramentas de cada máquina, pode ser gerado em tempo determinístico polinomial utilizando-se o método KTNS, conforme mostrado por Tang e Denardo (1988).

A solução do problema é composta pela lista de tarefas atribuídas e sequenciadas em cada máquina e o valor de *makespan* obtido. A Figura 4 demonstra uma solução para a instância exemplificada. As tarefas são representadas por retângulos preenchidos com larguras proporcionais ao tempo de processamento das mesmas. O tempo de troca de ferramentas, quando necessário, é representado por um retângulo hachurado de largura

proporcional ao tempo utilizado.

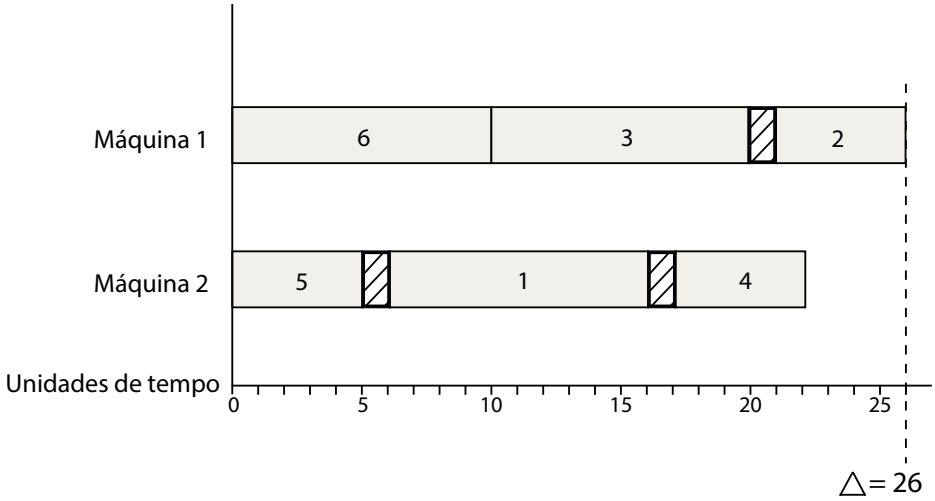


Figura 4 – Exemplo de solução para instância apresentada na Tabela 4.

No exemplo mostrado na Figura 4, foram alocadas à máquina 1 as tarefas [6, 3, 2]. O tempo de processamento da máquina 1 é a soma do tempo de processamento das tarefas a ela atribuídas [10, 10, 5], com o tempo gasto para as trocas de ferramentas $(1 \times \bar{p})$. Considera-se que o *magazine* da máquina 1 foi inicialmente preenchido com as ferramentas {1, 2, 3, 5, 6, 8}. Tais ferramentas tornam a máquina 1 apta a processar as tarefas 6 e 3, entretanto, antes que a tarefa 2 seja processada é necessário inserir no *magazine* a ferramenta 4, gerando uma troca de ferramenta. O tempo de processamento da máquina 1 é $10 + 10 + 5 + (1 \times \bar{p}) = 26$.

Para a máquina 2 são alocadas as tarefas [5, 1, 4] com os tempos de processamento [5, 10, 5]. Considera-se que o magazine da máquina 2 foi inicialmente preenchido com as ferramentas {3, 5, 6, 7, 8, 9}. Para o processamento sequencial das tarefas 1 e 4 é necessário incluir no *magazine* as ferramentas 1 e 10, gerando duas trocas de ferramentas. O tempo de processamento da máquina 2 é $5 + 10 + 5 + (2 \times \bar{p}) = 22$.

O *makespan* da solução é dado pelo tempo de processamento da máquina com maior tempo de processamento, que, no exemplo acima, corresponde ao tempo de processamento da máquina 1, ou seja, 26 unidades de tempo. Assim como no SSP, o plano de trocas de ferramentas pode ser representado por uma matriz binária. Para cada máquina i, temos uma permutação de colunas π_i e uma matriz binária $R_i^{\pi_i}$. O plano de troca de ferramentas de ambas as máquinas é apresentado na Tabela 5. As ferramentas removidas ou inseridas imediatamente antes do início do processamento de uma tarefa estão sublinhadas. A configuração inicial da máquina não é considerada no cálculo das trocas.

Uma solução representada pela matriz $R_i^{\pi_i}$ é avaliada pela Equação (2.1) proposta por Crama et al. (1994) para o SSP. Esta equação calcula o número de inversões de 0 para 1 na matriz. Tais inversões representam as inserções de ferramentas na máquina.

Tabela 5 – Planos de troca de ferramentas para máquinas paralelas.

(a) Máquina 1

Ferramentas	Tarefas		
	6	3	2
1	1	1	0
2	1	1	1
3	1	1	1
4	0	0	1
5	1	1	1
6	1	1	1
7	0	0	0
8	1	1	1
9	0	0	0
10	0	0	0

(b) Máquina 2

Ferramentas	Tarefas		
	5	1	4
1	0	0	1
2	0	0	0
3	1	1	1
4	0	0	0
5	1	1	1
6	1	1	0
7	1	1	1
8	1	0	1
9	1	1	1
10	0	1	1

A função objetivo do IPMTC é a minimização do *makespan*, resultado da Equação (2.3), que considera a variável $x_{ij} = 1$ caso a tarefa j seja processada pela máquina i, e $x_{ij} = 0$ caso contrário. A equação realiza a soma entre o tempo de processamento das tarefas alocadas à máquina i e a multiplicação entre o número de trocas de ferramentas do plano de produção com o tempo necessário para a troca de uma ferramenta.

$$\Delta = \max\left\{ \sum_{j \in J} x_{ij} p_j + Z_{SSP}(R_i^{\pi_i}) \times \overline{p} \right\}, \ \forall i \in M \tag{2.3}$$

De forma análoga ao SSP, o CBM aparece como um subproblema do IPMTC. Entretanto, dada a representação individual de cada máquina como uma matriz binária, temos a ocorrência de um CBM para cada máquina, o que reforça a importância deste problema no contexto de SMF.

2.2.1 Trabalhos relacionados

As bases para a formulação teórica do IPMTC foram apresentadas no trabalho de Stecke (1983). Em seu trabalho, o autor listou cinco objetivos gerais para maximizar a produção e reduzir custos em um plano de produção. Entre estes objetivos, encontram-se o agrupamento de máquinas em grupos com características idênticas e o escalonamento das tarefas e ferramentas. A combinação destes dois problemas fornece uma caracterização muito próxima à definição formal do IPMTC.

Posteriormente, Berrada e Stecke (1986) apresentam um algoritmo de *branch-and-bound* para o escalonamento de tarefas e ferramentas visando o balanceamento da carga de trabalho entre as máquinas de um mesmo grupo. Os experimentos computacionais utilizaram 55 instâncias contendo entre 4 e 48 tarefas. O método proposto mostrou-se

apto a apresentar soluções ótimas em baixo tempo computacional.

De acordo com Pinedo (2008), diversos ambientes de produção são generalizações do ambiente constituído por máquinas idênticas paralelas. Em um destes ambientes, o *flow shop* flexível, Koulamas (1991) apresenta dois algoritmos com o objetivo de minimizar o número total de trocas de ferramentas em cada nível do ambiente de produção sem aumentar o *makespan*. Além das trocas de ferramentas devido aos requisitos das tarefas, são consideradas também trocas de ferramentas devido ao fim da vida útil das mesmas. Os experimentos computacionais utilizaram dez instâncias com 4, 8 e 15 níveis de produção. O experimento forneceu soluções ótimas para todas as instâncias.

Abordando o *job shop* com restrições de ferramentas, outro ambiente que generaliza o ambiente constituído por máquinas idênticas paralelas, Hertz e Widmer (1996) apresentam uma busca tabu com o objetivo de minimizar o *makespan*. Os experimentos computacionais consideraram oito problemas com máquinas e tarefas variando entre 1 e 10. Os resultados obtidos foram comparados com os de uma versão anterior da busca tabu apresentada por Widmer (1991). Os resultados anteriores foram superados para a maioria das instâncias.

Agnetis, Nicolo e Lucertini (1991) também apresentam uma busca tabu com o objetivo de minimizar o *makespan*. Entretanto, o ambiente abordado é composto por apenas duas máquinas idênticas paralelas contendo *magazine* de capacidade igual a um. As ferramentas são compartilhadas entre as máquinas o que acrescenta uma restrição ao problema referente a disponibilidade das mesmas. Os autores comparam diferentes estratégias hierárquicas de decomposição e concluem que o escalonamento das tarefas deve ser priorizado enquanto o escalonamento das ferramentas pode ser considerado de forma subordinada.

Os trabalhos listados até aqui consideram a versão *offline* do IPMTC, ou seja, todas as tarefas são conhecidas *a priori* e estão disponíveis para escalonamento a partir do tempo zero. Kurz e Askin (2001) apresentam um trabalho que considera também a versão *online* do problema. Nesta versão, as tarefas não são conhecidas *a priori* e estão disponíveis para escalonamento em diferentes instantes de tempo. Ambas as versões são modeladas como TSP e resolvidas usando três diferentes heurísticas e um algoritmo genético. Os resultados obtidos são comparados apenas entre as versões, impossibilitando uma melhor análise do experimento.

A definição formal definitiva do IPMTC foi apresentada por Fathi e Barnette (2002). Neste trabalho os autores abordam o problema utilizando heurísticas baseadas em processamento de listas, procedimentos de busca local combinados e modelagem do problema como uma instância do TSP. Com o objetivo de minimizar o *makespan* foram realizados experimentos computacionais que compararam os resultados obtidos pelos diferentes métodos propostos e dois limites inferiores introduzidos no próprio trabalho. Os métodos baseados em procedimentos de busca local combinados reportaram os melhores

valores de solução. Os limitantes inferiores mostraram-se ineficazes, visto que para uma instância foi reportado um valor de solução superior ao reportado pelos métodos baseados em buscas locais.

Após um hiato de anos sem publicações específicas, Beezão et al. (2017) voltam a abordar o IPMTC propondo uma implementação da metaheurística *Adaptive Large Neighborhood Search* (ALNS) e dois modelos de programação linear inteira para a minimização do *makespan*. Os experimentos consideraram 2880 instâncias propostas no próprio artigo. As instâncias foram agrupadas em dois subconjuntos contendo 1440 instâncias cada. O subconjunto contendo os menores problemas possui instâncias que variam entre 8 e 25 tarefas. O subconjunto contendo os maiores problemas possui instâncias variando entre 50 e 200 tarefas. Considerando-se o limite de tempo estabelecido em 4200 segundos, os modelos de programação linear inteira mostraram-se ineficientes, reportando soluções ótimas apenas para as menores instâncias. Para comparar os resultados obtidos pela ALNS, os autores implementaram os métodos baseados em buscas locais combinadas introduzidos por Fathi e Barnette (2002). Embora à época não tenham sido publicados resultados individuais, os autores reportaram que o ALNS obteve melhores soluções para todas as instâncias.

Soares e Carvalho (2020) abordam o IPMTC propondo uma implementação da metaheurística BRKGA hibridizada com procedimentos de buscas locais. Entre as buscas locais propostas, novamente é utilizada a busca local por agrupamento de blocos de uns consecutivos (PAIVA; CARVALHO, 2017), aproveitando-se da relação existente entre o IPMTC e o CBM. Os experimentos computacionais realizados consideraram os dois conjuntos do único *benchmark* de instâncias disponível na literatura (BEEZÃO et al., 2017). Para o conjunto contendo os menores problemas, os resultados reportados pelo método proposto foram comparados a um limitante superior obtido pela implementação de um modelo de programação linear inteira mista proposto por Beezão et al. (2017). O método proposto reportou soluções melhores ou iguais para 99,86% das instâncias deste conjunto. Para o conjunto contendo os maiores problemas, os resultados obtidos pelo método proposto foram comparados aos resultados originais reportados pela ALNS de Beezão et al. (2017). O BRKGA reportou melhores valores de solução para 93,89% das instâncias deste conjunto. Além da maior qualidade nas soluções reportadas, o tempo de execução do BRKGA foi, em média e após uma ponderação, 96,50% menor do que o ALNS. Diante do exposto, o trabalho de Soares e Carvalho (2020) constitui o atual estado da arte para o IPMTC.

Uma variação da versão *online* do IPMTC foi abordada por Dang et al. (2021). Neste trabalho, em vez de trocas individuais, todas as ferramentas carregadas no *magazine* da máquina são trocadas a cada operação de configuração. Com o objetivo de minimizar o atraso total e o número de trocas de ferramentas, os autores apresentam um modelo de

programação linear inteira mista e um algoritmo genético. Os experimentos computacionais realizados utilizaram instâncias reais e compararam o modelo proposto, o algoritmo genético e uma heurística projetada para simular o escalonamento manual praticado pela indústria que forneceu as instâncias. O modelo proposto mostrou-se incapaz de produzir soluções factíveis para a maioria das instâncias. Os resultados reportados pelo algoritmo genético apresentaram uma melhoria média entre 20% e 40% quando comparado ao escalonamento manual.

3 Minimização de blocos de uns consecutivos

Este capítulo é fortemente baseado no artigo *"Heuristic methods to consecutive block minimization"* de autoria de Leonardo Cabral da Rocha Soares, Jordi Alves Reinsma, Luis Henrique Leão do Nascimento e Marco Antonio Moreira de Carvalho, publicado no periódico *Computers & Operations Research*, Volume 120 de agosto de 2020. O artigo aborda o problema de minimização de blocos de uns, apresentando uma nova representação em grafos para o problema, um método heurístico, uma abordagem metaheurística e um método exato baseado na redução do problema para uma versão particular do TSP. O método exato é de autoria exclusiva de Jordi Alves Reinsma e Marco Antonio Moreira de Carvalho, tendo sido desenvolvido em um projeto separado. Sua inclusão no artigo deu-se a partir do processo de revisão por pares, como solicitação de um dos revisores. Diante do exposto, a seção referente ao método exato foi transcrita no Anexo A. A seguir, são apresentados os fundamentos teóricos do problema abordado, os trabalhos relacionados, os métodos computacionais propostos, à exceção do mencionado método exato, os experimentos realizados e as conclusões.

Diversos problemas estudados em otimização discreta podem ser modelados com a utilização de matrizes binárias. Esta estratégia é extremamente comum quando aborda-se sistemas de equações, equações em duas dimensões ou relações de pertinência. Alguns destes problemas são oriundos de ambientes industriais, tais como o problema de determinação de leiaute de matrizes porta (ou *Gate Matrix Layout Problem*, GMLP) e o SSP (LINHARES; YANASSE, 2002).

Os problemas mencionados são intrinsecamente permutacionais, ou seja, sua solução é determinada a partir do ordenamento das colunas (ou linhas) em uma matriz binária. Embora a avaliação da solução seja dependente do problema, em geral, ela considera a minimização das descontinuidades em uma matriz binária. Por exemplo, no GMLP, as colunas da matriz binária representam as portas e as linhas as redes que as conectam. Elementos não nulos na matriz binária indicam que um transistor está implementando em uma porta e rede específicas. Todos os transistores em uma mesma rede devem ser conectados entre si. A solução para o problema consiste em uma permutação de colunas que minimize o comprimento dos fios utilizados para conectar as redes.

Para o SSP, as colunas da matriz binária representam as tarefas escalonadas para a máquina e as linhas representam as ferramentas disponíveis. A existência de um elemento não nulo na matriz binária indica que uma determinada ferramenta é necessária para o processamento de uma determinada tarefa. A inversão dos valores dos elementos em uma mesma linha da matriz binária indica uma troca de ferramenta. A solução para o problema

consiste em uma permutação de colunas que minimize o número de trocas de ferramentas.

Em uma matriz binária, uma sequência contígua de elementos não nulos é denominada de *blocos de uns*. A existência de qualquer quantidade de elementos nulos entre dois blocos de uns caracteriza uma descontinuidade. A minimização de blocos de uns consecutivos (ou *Consecutive Block Minimization*, CBM) é um problema $\mathcal{NP}$-Difícil (GAREY; JOHNSON, 1979b; KOU, 1977) definido como o problema de encontrar uma permutação de colunas em uma matriz binária que minimize o número total de blocos de uns.

Formalmente, o CBM considera uma matriz binária $A = \{a_{ij}\}$ com m linhas e n colunas, sendo $a_{ij} \in \{0, 1\}$, $i = \{1, 2, ..., m\}$, and $j = \{1, 2, ..., n\}$. Caso exista uma permutação de colunas que acarrete na existência de apenas um bloco de uns por linha, a matriz possui uma característica conhecida como propriedade dos uns consecutivos (ou Consecutive Ones Property, C1P), introduzido por Fulkerson e Gross (1965). Entretanto, tal característica raramente é encontrada.

Como ilustração, a matriz binária $A_{4,5}$ é apresentada como uma instância hipotética do CBM. A primeira linha possui três blocos de uns, ocasionados pelas descontinuidades geradas pelos elementos $a_{1,2}$ e $a_{1,4}$. A segunda linha possui três blocos de uns, ocasionados pelas descontinuidades em $a_{2,2}$ e $a_{2,4}$. Na terceira linha, a descontinuidade em $a_{3,3}, a_{3,4}$ resulta em dois blocos de uns. Finalmente, a quarta linha possui dois blocos de uns ocasionados pela descontinuidade em $a_{4,3}$. No total, esta instância hipotética possui dez blocos de uns. Para facilitar a identificação as descontinuidades mencionadas são apresentadas sublinhadas na referida matriz.

$$A = \begin{bmatrix} 1 & \underline{0} & 1 & \underline{0} & 1 \\ 1 & \underline{0} & 1 & \underline{0} & 1 \\ 1 & 1 & \underline{0} & \underline{0} & 1 \\ 0 & 1 & \underline{0} & 1 & 0 \end{bmatrix}$$

O objetivo do CBM é gerar uma permutação de colunas que minimize a quantidade de blocos de uns. Assim, a solução para a instância representada pela matriz A é uma permutação π de suas colunas. A matriz binária A^{π_1} mostra uma possível solução onde as colunas da matriz A foram ordenadas de acordo com a permutação $\pi_1 = [2, 1, 3, 4, 5]$. As duas primeiras colunas, exibidas em negrito na matriz A^{π_1}, tiveram suas posições alternadas. Como resultado, o número de blocos de uns nas duas primeiras linhas foi reduzido para dois. As linhas três e quatro não sofreram alterações no número de blocos de uns. O número total de blocos de uns foi reduzido para oito.

$$A^{\pi_1} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & 1 & 0 & 1 \\ \mathbf{0} & \mathbf{1} & 1 & 0 & 1 \\ \mathbf{1} & \mathbf{1} & 0 & 0 & 1 \\ \mathbf{1} & \mathbf{0} & 0 & 1 & 0 \end{bmatrix}$$

Embora a permutação π_1 forneça uma solução válida para a instância apresentada, ela não representa uma solução ótima. Para a matriz apresentada, que possui a característica C1P, a solução ótima é dada pela permutação $\pi_2 = [3, 1, 5, 2, 4]$, contendo quatro blocos de uns. A matriz A^{π_2} representa esta solução.

$$A^{\pi_2} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

O número de bloco de uns de uma solução representada pela matriz de permutação A^{π} é calculada pela Equação (3.1). Proposta por Crama et al. (1994) para o SSP, esta equação calcula o número de inversões de zero para um. Apesar do número de inversões ser maior do que o número de blocos de uns, ambas medidas são proporcionais, assim, a equação pode ser utilizada para mensurar a qualidade da solução. Para que os blocos de uns iniciados a partir da primeira colunas sejam considerados pela Equação (3.1), uma coluna adicional fictícia contendo apenas elementos com valor igual a zero deve ser anexada à matriz, ou seja, $A^{\pi}_{i0} = 0$.

$$Z^{\pi}_{CBM}(A^{\pi}) = \sum_{j=1}^{n} \sum_{i=1}^{m} A^{\pi}_{ji}(1 - A^{\pi}_{ji-1}) \tag{3.1}$$

Conforme já mencionado, o objetivo do CBM é a minimização do número de blocos de uns em uma matriz binária. Assim, é possível definir sua função objetivo com a Equação (3.2). A equação visa determinar a permutação π das colunas da matriz A, entre o conjunto Π de todas as permutações possíveis, que minimize o número de blocos de uns.

$$\min_{\pi \in \Pi} \left\{ Z^{\pi}_{CBM}(A) \right\} \tag{3.2}$$

3.1 Trabalhos relacionados

Problemas envolvendo blocos de uns consecutivos em matrizes binárias são encontrados na literatura desde a década de 1950. Dom (2009) apresenta uma revisão abrangente sobre a literatura envolvendo C1P e blocos de uns consecutivos em matrizes binárias. Kou (1977) e Garey e Johnson (1979b) demonstraram que o CBM pertence à classe de

problemas $\mathcal{NP}$-Difícil. Em um trabalho posterior, Haddadi (2002) demonstrou que esta classificação permanece mesmo em um caso restrito, em que as matrizes binárias possuem exatamente dois blocos de uns consecutivos por linha.

Apesar da grande aplicação prática e alta relevância teórica, a literatura específica sobre o CBM é muito restrita. Excetuando-se o artigo que fornece as bases para este capítulo, não há menção na literatura sobre abordagens utilizando-se métodos exatos ou metaheurísticas. Entre as abordagens encontradas na literatura, assim como ocorre para o SSP, a modelagem do problema como uma instância do TSP é recorrente (DYSON; GREGORY, 1974; KOU, 1977; MADSEN, 1988; ALIZADEH et al., 1995; JOHNSON et al., 2004; HADDADI; LAYOUNI, 2008). Para que tal modelagem seja viável, faz-se necessário estabelecer uma noção de distância entre as colunas do CBM para a criação da matriz de distância do TSP. Entre as métricas utilizadas neste contexto, destaca-se a *distância de Hamming*. Em geral, após a modelagem o sequenciamento das colunas é obtido utilizando-se alguma heurística específica para o TSP, como a heurística do vizinho mais próximo, ou um algoritmo de aproximação, como o algoritmo de Christofides (CHRISTOFIDES, 1976).

Ao abordarem o problema de corte de estoque (ou *Cutting-stock Problem*, CSP) em uma indústria de vidros, Dyson e Gregory (1974) modelaram um subproblema de alocação de padrões como uma instância do CBM. Uma matriz binária foi utilizada para indicar quais padrões de corte produziam cada tipo de peça de vidro. O objetivo do trabalho era minimizar as descontinuidades na produção das peças, visto que tais descontinuidades poderiam acarretar diferenças nos tons das peças produzidas. O CSP foi resolvido via programação linear e o subproblema de alocação de padrões foi modelado como TSP. Entretanto, os recursos computacionais disponíveis na época não possibilitaram a resolução sequer das menores instâncias. Assim, esta modelagem foi descartada e um algoritmo de *branch-and-bound* modificado foi proposto. Embora os resultados reportados sejam melhores do que o escalonamento manual, o alto custo de implantação de sistemas computacionais na época inviabilizava a proposta.

O CBM é novamente abordado como um subproblema de alocação de padrões por Madsen (1979). Neste trabalho, o CBM é nomeado como problema de minimização de descontinuidades. Os padrões de corte foram obtidos resolvendo-se o problema da mochila e uma heurística originalmente aplicada ao problema de largura de banda em matrizes foi aplicada para reduzir a distância de produção entre as peças de um mesmo tipo, e, consequentemente, minimizar as descontinuidades. Em um estudo posterior abordando o mesmo contexto, Madsen (1988) modelou o CBM como uma instância do TSP e resolveu a modelagem utilizando uma heurística subótima proposta por Lin (1965).

Alizadeh et al. (1995) modelaram um problema relacionado ao sequenciamento de DNA como uma instância do CBM. Novamente, o problema foi modelado como TSP e

resolvido heuristicamente. A matriz de distâncias foi elaborada com base na distância de Hamming entre as colunas do CBM, o que posteriormente mostrou-se como parte de um mapeamento perfeito do CBM para o TSP (ATKINS; BOMAN; HENDRICKSON, 1998).

Johnson et al. (2004) abordam o CBM no contexto de compressão de dados. As matrizes binárias originais são particionadas em submatrizes menores e modeladas como instâncias do TSP. O TSP é resolvido heuristicamente e as soluções combinadas para reconstituir soluções para as matrizes originais.

Haddadi e Layouni (2008) abordam o CBM visando reduzir a complexidade de resolução de sistemas de equações lineares. Utilizando reduções entre problemas, os autores apresentam um algoritmo de aproximação de fator 1,5. A matriz binária é abordada como uma instância do problema de minimização de blocos circulares (HSU; MCCONNELL, 2003). Neste problema, as colunas e os blocos de uns consecutivos são consideradas de forma circular e o problema é transformado em uma instância simétrica do TSP. A matriz de distâncias é calculada utilizando-se a distância de Hamming, o que caracteriza o caixeiro viajante de Hamming (*Hamming traveling salesman problem*, HTSP), introduzido por Reinelt (1994). Após as transformações, os autores utilizam o algoritmo de Christofides (CHRISTOFIDES, 1976) para solucionar as instâncias. A transformação do CBM para o HTSP é descrita e exemplificada no Anexo A.

Ainda no contexto de redução de dimensões de sistemas de equações lineares, Haddadi et al. (2015) apresentam um método heurístico baseado em dois procedimentos de busca local para o CBM. A primeira busca local altera a posição de uma coluna na solução, removendo a mesma de sua posição original e a reinserindo em uma posição diferente. A segunda busca local troca as posições de duas colunas não adjacentes na solução. Ambas as buscas locais exploram todas as permutações existentes em suas respectivas vizinhanças. Sempre que uma melhor solução é encontrada, a exploração da vizinhança é reiniciada a partir desta nova solução. O procedimento de busca local encerra-se quando a exploração da vizinhança atual não acarretar em melhoria da solução. O algoritmo proposto, a partir de uma solução inicial gerada aleatoriamente, aplica a primeira busca local até que um ótimo local seja encontrado. A segunda busca local é então aplicada a partir da solução obtida pelo procedimento anterior. O método termina quando um ótimo local é encontrado na segunda vizinhança.

Além da heurística proposta, Haddadi et al. (2015) apresentam um procedimento de avaliação rápida, ou δ-avaliação, que permite que o efeito dos movimentos das buscas locais sejam avaliados sem a necessidade de reavaliação da solução inteira. Enquanto a avaliação completa de uma solução possui complexidade assintótica limitada por $\Theta(mn)$, a δ-avaliação possui complexidade limitada por $\mathcal{O}(m)$. Os experimentos computacionais realizados utilizaram 5 instâncias reais de uma empresa ferroviária alemã e 45 instâncias artificiais geradas aleatoriamente no próprio trabalho. Este é o único conjunto de instâncias

benchmark disponível na literatura e os resultados reportados pelo método proposto constituem o estado da arte para o CBM até então. Por este motivo, estes resultados e instâncias são considerados nos experimentos relatados na Seção 3.3.

3.2 Métodos

Esta seção descreve em detalhes as novas contribuições propostas na abordagem ao CBM: (i) uma representação em grafos; (ii) uma heurística baseada em grafos; (iii) o uso de um procedimento de busca local que considera as características intrínsecas do CBM; e (iv) uma implementação de uma metaheurística busca local iterada (ou *iterated local search*, ILS).

3.2.1 Uma nova representação em grafos

Ao tratarmos de matrizes de entrada do CBM, a abordagem padrão dá-se a partir da avaliação das colunas da mesma. Buscando fugir deste padrão, apresenta-se um grafo que possibilita a análise das linhas da matriz em busca de informações que possam ser utilizadas como regras heurísticas. Neste grafo, os vértices representam as linhas da matriz binária, as arestas conectam os elementos não nulos em cada coluna e os pesos das arestas são determinados pela frequência com que isto ocorre. Diferente das abordagens tradicionais, a ocorrência dos elementos não nulos é considerada por linha e não por coluna. Este grafo é semelhante ao utilizado em Carvalho e Soma (2015), Paiva e Carvalho (2017), Lima e Carvalho (2017) e ao grafo de intersecção introduzido por Fulkerson e Gross (1965). Entretanto, os pesos das arestas aqui propostos são específicos para o CBM.

Para exemplificar, considere a matriz binária A que representa uma instância do CBM. A matriz é composta por quatro linhas, portanto, o grafo CBM possui quatro vértices. A primeira coluna da matriz A possui elementos não nulos nas linhas 1, 2 e 3. Assim, as arestas $\{1, 2\}$, $\{1, 3\}$ e $\{2, 3\}$ são adicionadas ao grafo. Como estas arestas não existiam anteriormente, o peso delas é definido como um. A segunda coluna possui elementos não nulos nas linhas 3 e 4. Como o grafo ainda não possui uma aresta entre estes vértices, a aresta $\{3, 4\}$ é adicionada com peso igual a um. A terceira coluna possui elementos não nulos nas linhas 1 e 2. A aresta $\{1, 2\}$ já foi adicionada ao grafo, assim, seu peso á atualizado para dois. A quarta coluna possui elementos não nulos apenas na 4 linha, assim, nenhuma aresta é adicionada ou atualizada no grafo. A quinta coluna possui elementos não nulos as linhas 1, 2 e 3. Todas as arestas correspondentes já foram adicionadas ao grafo, portanto, a aresta $\{1, 2\}$ tem seu peso atualizado para três e as arestas $\{1, 3\}$ e $\{2, 3\}$ têm seus pesos atualizados para dois. O grafo gerado é ilustrado pela Figura 5.

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

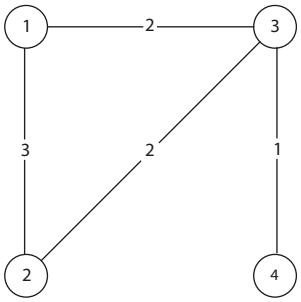


Figura 5 – Grafo CBM.

3.2.2 Heurística baseada em grafos

A representação em grafos proposta oferece uma perspectiva diferente do CBM. No lugar da abordagem tradicional que analisa as colunas, esta representação oferece uma perspectiva relacionada as linhas da matriz, fornecendo *insights* úteis para resolução do CBM. Assim, apresenta-se uma nova heurística gulosa baseada no percurso da representação em grafos proposta. A ideia principal é obter um sequenciamento de linhas e posteriormente utilizar esta informação para se obter a solução do CBM, ou seja, um sequenciamento de colunas. Esta estratégia tem sido empregada com sucesso em diferentes problemas que utilizam matrizes binárias em suas abstrações (CARVALHO; SOMA, 2015; PAIVA; CARVALHO, 2017; LIMA; CARVALHO, 2017). Para o percurso do grafo, utiliza-se um algoritmo de busca em largura (ou *breadth-first search*, BFS) com algumas adaptações.

Dado um vérticie inicial, a BFS explora todos os vértices vizinhos, isto é, os vértices com os quais este vértice compartilha uma aresta. Após isso, para cada vértice vizinho, a BFS explora toda a sua vizinhança. Cada vértice é explorado uma vez e o processo termina quando todo o grafo tiver sido explorado. Caso o grafo possua mais de um componente, após a exploração completa do componente que contém o vértice inicial, a BFS seleciona um vértice presente em outro componente e reinicia a exploração. Assim, grafos desconectados são processados naturalmente. A BFS retorna uma lista ϕ contendo a ordem em que todos os vértices presentes no grafo foram explorados.

A primeira modificação proposta para a BFS é o mecanismo de escolha do vértice inicial. A implementação proposta seleciona o vértice de menor grau como vértice inicial. Esta escolha é feita visando evitar que este vértice, que representa a linha com menos elementos relacionados, seja escalonado no meio de uma solução, pois pretende-se que as

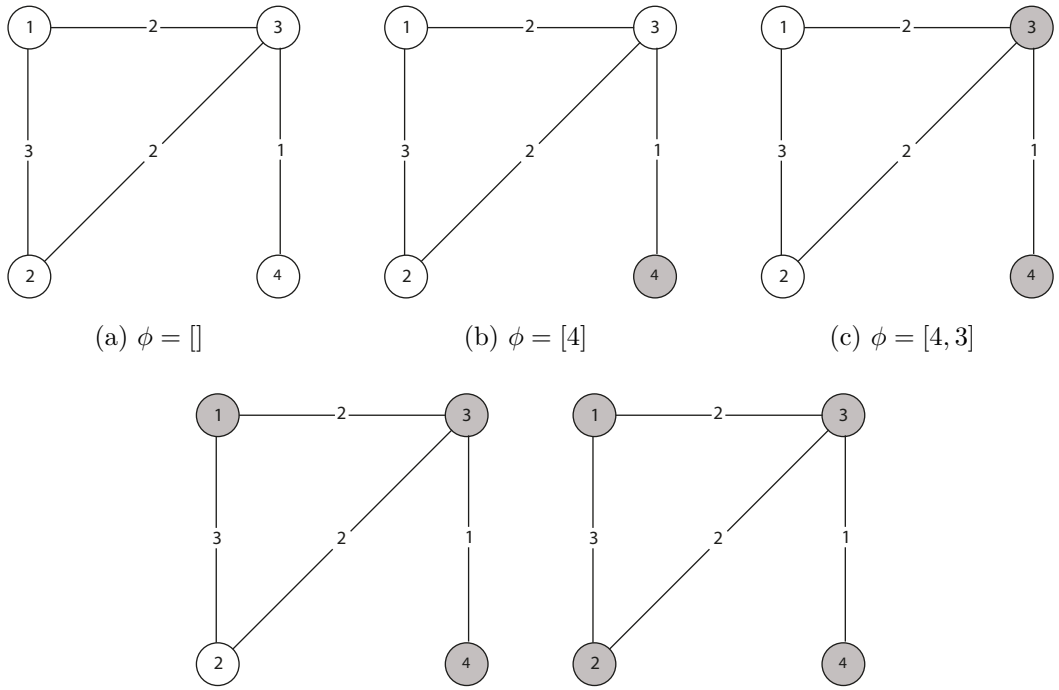
linhas fortemente relacionadas sejam escalonadas de forma contígua. Eventuais empates são tratados a partir da seleção dos vértices em ordem lexicográfica. A segunda modificação refere-se à ordem de exploração dos vértices. Em uma vizinhança, um vértice tem prioridade proporcional ao peso da aresta que o conecta ao vértice cuja vizinhança está sendo explorada. Novamente, os empates são resolvidos selecionando-se os vértices em ordem lexicográfica. O objetivo deste critério guloso é sequenciar elementos que frequentemente aparecem juntos e posteriormente usar esta informação na geração da permutação das colunas.

As etapas da BFS adaptada são apresentadas no Algoritmo 1. Um grafo CBM $G = (V, E)$ é fornecido como parâmetro de entrada. Uma fila vazia é criada para guiar o processo de busca e a lista ϕ é inicializada como vazia (linhas 2 e 3). O laço principal (linhas 4 a 17) garante que todos os vértices em V sejam explorados. O vértice i com o menor grau é selecionado como vértice inicial (linha 5), adicionado à lista ϕ (linha 6), à fila Q (linha 7) e removido do conjunto de vértices V (linha 8). O laço interno (linhas 9 a 16) garante a exploração de cada componente de G. O vértice v, no início da fila, é removido (linha 10) e cada vértice em sua vizinhança, denotada por $N(v)$, é removido do grafo e adicionado a Q e ϕ (linhas 12 a 14). Após a exploração de todos os vértices, o algoritmo termina e a lista ϕ é retornada (linha 18). Todos os vértices são explorados, assim, o algoritmo possui complexidade limitada por $\mathcal{O}(|V| + |E|)$ utilizando uma representação por listas de adjacências.

Algoritmo 1: BFS adaptado.

```
   Entrada: Grafo CBM G = (V, E);
   Saída: Lista de vértices φ;
1  início
2      crie uma fila vazia Q;
3      φ ← ∅;
4      enquanto V ≠ ∅ faça
5          i ← o vértice de menor grau V;
6          insira i ao final de φ;
7          Q ← i;
8          V \ i;
9          enquanto Q ≠ ∅ faça
10             v ← Q.desenfileirar();
11             para cada w ∈ N(v) faça
12                 V \ w;
13                 Q ← w;
14                 insira w ao final φ;
15             fim
16         fim
17     fim
18     retorna φ;;
19 fim
```

A Figura 6 ilustra a aplicação da BFS adaptada ao grafo CBM previamente apresentado na Figura 5. Os vértices já explorados são destacados em cinza. Inicialmente, como demonstrado em (a), nenhum vértice foi explorado e ϕ está vazia. Em (b), o vértice 4 é selecionado por ser o vértice de menor grau no grafo. Este vértice é adicionado a ϕ e sua vizinhança é explorada. Há somente um vértice vizinho, o vértice 3. Assim, este vértice é explorado e adicionado a ϕ, conforme visto em (c). O vértice 3 possui dois vizinhos, os vértices 1 e 2. Ambos possuem o mesmo grau e o empate é resolvido pela exploração em ordem lexicográfica. Assim, o vértice 1 é adicionado a ϕ (d) e, posteriormente, o vértice 2 também é adicionado a ϕ (e). Como não há mais vértices a serem explorados, a BFS termina e a lista $\phi = [4, 3, 1, 2]$ é retornada.



(a) $\phi = []$ (b) $\phi = [4]$ (c) $\phi = [4, 3]$

(d) $\phi = [4, 3, 1]$ (e) $\phi = [4, 3, 1, 2]$

Figura 6 – BFS adaptada aplicada a um grafo CBM.

Conforme mencionado, a BFS gera uma lista ordenada ϕ de linhas da matriz de entrada. Entretanto, uma solução factível para o CBM consiste em uma permutação π de colunas da matriz de entrada. Assim, é necessário gerar tal permutação a partir de ϕ. Becceneri, Yanasse e Soma (2004) apresentam um método guloso para este propósito no contexto do problema de minimização de pilhas abertas, um problema relacionado ao CBM. O método percorre a lista de linhas sequenciando as colunas. Uma coluna é inserida na solução π uma vez que as linhas de seus elementos não nulos corresponderem às linhas percorridas em ϕ. Se mais de uma coluna atender a este critério, a ordem lexicográfica é adotada.

As etapas do método guloso para sequenciamento de colunas são apresentadas no

Algoritmo 2. A lista de linhas ϕ e a matriz binária A_{mn} do CBM são fornecidas como parâmetros de entrada. Inicialmente, a permutação de colunas π e o conjunto auxiliar de linhas R estão vazios (linhas 2 e 3). O laço principal (linhas 3 a 10) adiciona cada elemento de ϕ a R (linha 4). Então, um segundo laço (linhas 5 a 9) verifica se algum subconjunto de R representa os índices de linha de elementos não nulos de qualquer coluna em A. Uma função auxiliar $c(j)$ retorna os índices das linhas dos elementos não nulos da coluna j. Se o critério for atendido (linha 6) e se a coluna ainda não tiver sido adicionada a π, a coluna será adicionada ao final dela (linha 7). Depois que todas as colunas são inseridas na solução, o algoritmo termina e a permutação π é retornada (linha 11). O sequenciamento guloso de colunas requer o exame de toda a matriz binária. Portanto, o procedimento possui complexidade limitada por $\mathcal{O}(mn)$.

Algoritmo 2: Sequenciamento guloso de colunas

Entrada: lista de linhas ϕ, Matriz binária A;
Saída: permutação de colunas π;

```
1  π ← ∅;
2  R ← ∅;
3  para cada i ∈ φ faça
4  |   R ← R ∪ i;
5  |   para cada coluna j ∈ A faça
6  |   |   se c(j) ⊆ R and j ∉ π então
7  |   |   |   adicione j ao final de π;
8  |   |   fim
9  |   fim
10 fim
11 retorne π;
```

Consideremos a aplicação do Algoritmo 2 sobre a lista ϕ retornada pela aplicação da BFS no exemplo ilustrado pela Figura 6. Analisando a matriz A temos $c(1) = \{1, 2, 3\}$, $c(2) = \{3, 4\}$, $c(3) = \{1, 2\}$, $c(4) = \{4\}$ e $c(5) = \{1, 2, 3\}$. Percorrendo $\phi = [4, 3, 1, 2]$, a coluna 4 é sequenciada após a linha 4; a coluna 2 é sequenciada após a linha 3; nenhuma coluna é sequenciada após a linha 1 e as colunas 1, 3 e 5 são sequenciadas em ordem lexicográfica após a linha 2. A permutação $\pi = [4, 2, 1, 3, 5]$ é a solução obtida pelo sequenciamento guloso de colunas. A matriz A^π é obtida usando a permutação de colunas π na matriz A. Comparando-se A e A^π, o número de blocos de uns consecutivos foi reduzido de 10 para 5.

$$A^\pi = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

3.2.3 Busca local iterada

Diversos métodos metaheurísticos exploram o espaço de solução coordenando procedimentos de *intensificação* e *diversificação* até que um critério de parada seja alcançado. De forma geral, na etapa de intensificação é empregado algum procedimento de busca local que aplica sistematicamente um *movimento*, ou seja, um tipo específico de modificação em uma solução s, gerando uma nova solução s'. A região do espaço de busca alcançável pela aplicação de um movimento em particular é denominada *vizinhança* e a solução s' é denominada *solução vizinha* de s. Se s' é a melhor solução em uma dada vizinhança, ela é considerada um *ótimo local*. Entretanto, um *ótimo global*, isto é, um ótimo local comum a todas as vizinhanças, pode estar localizado em outra região do espaço de busca. Para diversificar as soluções encontradas, um mecanismo de *perturbação* é empregado. Tal mecanismo ocasiona um certo nível de desordem na solução, permitindo que outras regiões do espaço de soluções, antes inacessíveis por uma busca local, sejam exploradas. O equilíbrio entre intensificação e diversificação é uma abordagem para evitar que o processo de busca fique estagnado em ótimos locais.

A metaheurística ILS (LOURENÇO; MARTIN; STÜTZLE, 2019) realiza uma exploração enviesada do espaço de soluções. A partir de uma dada solução inicial s para um problema de otimização específico, a ILS aplica alternadamente mecanismos de intensificação e diversificação, gerando uma nova solução s' que é utilizada como solução inicial para a próxima iteração do método. Apesar de conceitualmente simples, a ILS tem sido aplicada com sucesso a diversos problemas computacionalmente difíceis (LOURENÇO; MARTIN; STÜTZLE, 2019).

Considerando-se as características da ILS, apresenta-se uma implementação da mesma aplicada ao CBM. Para tanto, a solução inicial é construída utilizando-se a nova representação de grafos apresentada e a heurística construtiva proposta, ambos descritos anteriormente. Considerando-se as características do CBM, na etapa de intensificação são aplicados o método *2-opt* (CROES, 1958) e a busca local por agrupamento de blocos de uns, proposta por Paiva e Carvalho (2017). Na etapa de diversificação, é aplicado o método *2-swap*. Além disso, visando um equilíbrio entre o tempo de execução e a convergência da solução, a aplicação da perturbação foi limitada a uma porcentagem α do tamanho da solução. Os critérios de parada utilizados e o valor de α foram definidos nos experimentos preliminares descritos na Seção 3.3.

As etapas da ILS proposta são descritas no Algoritmo 3. O parâmetro α é fornecido como um parâmetro de entrada. A solução inicial π é gerada utilizando-se a BFS adaptada sobre a nova representação em grafo para o CBM e a heurística gulosa para o sequenciamento de colunas (linha 1). O laço principal (linhas 2 a 9) garantem a execução da ILS enquanto um critério de parada não for encontrado. A solução atual s sofre uma perturbação. Todos os pares possíveis de índices de colunas são embaralhados e $\alpha/2$ desses

pares são selecionados para troca, gerando-se uma solução α-vizinha π' (linha 3). Em seguida, a busca local é realizada aplicando-se sequencialmente os métodos 2-opt (linha 4) e agrupamento de blocos de uns (linha 5). Após este processo de intensificação, a solução π' é um ótimo local para ambas as vizinhanças consideradas. Caso o valor de avaliação de π' seja melhor do que o valor de avaliação de π, π' é aceito como nova solução atual (linhas 6 e 7) e será utilizada como solução inicial para a próxima iteração do método. Caso contrário, π' é descartada e π permanece como solução atual. Após atingir algum critério de parada, a solução atual π é retornada (linha 10) e o método termina.

Algoritmo 3: Iterated local search

Entrada: α;
Saída: solution π;
1 $\pi \leftarrow$ soluçãoInicial();
2 **enquanto** *critério de parada não for encontrado* **faça**
3 $\quad \pi' \leftarrow$ troque α colunas aleatórias de π;
4 $\quad$ **aplique** a busca local 2-opt a π';
5 $\quad$ **aplique** a busca local por agrupamento de blocos de uns a π';
6 $\quad$ **se** π' *é melhor que* π **então**
7 $\quad\quad \pi \leftarrow \pi'$;
8 $\quad$ **fim**
9 **fim**
10 **retorne** π;

Os componentes de diversificação e intensificação da metaheurística ILS, assim como o procedimento de avaliação rápida para o CBM, são descritos a seguir.

3.2.3.1 Método 2-swap

No bem conhecido método *2-swap*, um movimento consiste na troca de dois elementos de uma dada solução. Todos os $\mathcal{O}(n^2)$ possíveis pares de índices de colunas são gerados e embaralhados. Então, $\frac{\alpha}{2}$ destes pares são selecionados para a troca de suas posições na solução.

Como exemplo, consideremos a solução $\pi = [1, 2, 3, 4, 5]$ ilustrada pela matriz A^{π} contendo cinco blocos de uns consecutivos. Um par de elementos desta solução é aleatoriamente selecionado para troca, neste caso específico, as colunas 2 e 4 destacadas em cinza na matriz. Então, o método *2-swap* troca a posição dos elementos selecionados gerando a permutação $\pi' = [1, 4, 3, 2, 5]$, como ilustrado na matriz $A^{\pi'}$ contendo sete blocos de uns consecutivos.

$$A^{\pi} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad A^{\pi'} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

3.2.3.2 Método 2-opt

Um movimento no método 2-opt, originalmente proposto por Croes (1958) para o TSP, consiste em inverter as posições dos elementos de uma permutação dentro de um determinado intervalo. Todos os $\mathcal{O}(n^2)$ possíveis pares de índices de colunas são gerados e embaralhados. Em seguida, o movimento é avaliado, sequencialmente, para cada par de índices na solução. Em caso de melhoria na solução, o movimento é mantido e o procedimento reiniciado. Caso contrário, o movimento é descartado e o próximo par avaliado. O procedimento termina quando nenhum movimento resultar em melhoria da solução.

Como exemplo, consideremos a solução $\pi = [1, 2, 3, 4, 5]$ e o intervalo fechado $(2, 5)$. A matriz A^{π} representa a solução inicial possuindo cinco blocos de uns consecutivos. As colunas no intervalo selecionado estão destacadas em cinza. Um movimento do método 2-opt inverte a posição dos elementos no intervalo, gerando a solução $\pi' = [1, 5, 4, 3, 2]$ exibida na matriz $A^{\pi'}$ e que possui seis blocos de uns consecutivos.

$$A^{\pi} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad A^{\pi'} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2.3.3 Agrupamento de blocos de uns

Paiva e Carvalho (2017), abordando o SSP, apresentaram a busca local por agrupamento de blocos de uns com o objetivo de reduzir o número de descontinuidades em uma matriz binária. Para isso, a busca local percorre a matriz binária, linha por linha, em ordem aleatória, procurando por dois ou mais blocos de uns consecutivos. Para cada par de bloco de uns consecutivos, a busca local tenta mover as colunas que constituem o primeiro bloco de uns, uma a uma, para a esquerda e para a direita do segundo bloco de uns, na tentativa de agrupá-los. Cada movimento é avaliado e o movimento que resultar no melhor valor de solução é realizado. Todos os movimentos que não resultam em melhoria da solução são descartados. O procedimento se encerra quando todas as linhas da matriz tiverem sido analisadas. A análise dos blocos de uns é executada em tempo $\mathcal{O}(mn)$. A

avaliação do impacto de um determinado movimento na solução pode ser realizada em tempo $\mathcal{O}(m)$ utilizando um procedimento de avaliação rápida, conforme descrito a seguir. Assim, a complexidade desta busca local é limitada por $\mathcal{O}(m^2n)$.

Um movimento hipotético desta busca local é ilustrado pelas matrizes (a) a (d) abaixo. Em (a), temos uma matriz binária contendo cinco blocos de uns consecutivos. Em (b), a segunda linha é aleatoriamente selecionada para análise. A linha possui dois blocos de uns, um na primeira coluna e o outra na terceira coluna. A busca local avalia a movimentação da coluna contendo o primeiro bloco de uns para as posições à esquerda e à direita do segundo bloco de uns. O movimento à esquerda (c) acarreta em diminuição do número total de blocos de uns para quatro. O movimento à direita (d), acarreta em aumento no número de blocos de uns para seis. Assim, o movimento à esquerda resulta no melhor valor de solução e é executado.

(a) $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ (b) $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

(c) $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ (d) $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

3.2.3.4 Procedimento de avaliação rápida

A avaliação completa de uma solução para o CBM possui complexidade limitada por $\Theta(mn)$, em que m representa o número de linhas e n representa o número de colunas da matriz de entrada. Realizar tal avaliação para cada possível movimento da busca local, a cada iteração da ILS é computacionalmente dispendioso. Visando reduzir o esforço computacional e melhorar a performance da busca local, optou-se por utilizar a δ-avaliação proposta por Haddadi et al. (2015) para o CBM. Ao invés de avaliar toda a matriz, este procedimento de avaliação rápida analisa somente o impacto decorrido da mudança de posicionamento de uma coluna na matriz e possui complexidade limitada por $\mathcal{O}(m)$.

Especificamente, o número de blocos de uns consecutivos criados pela inserção de uma coluna j entre duas colunas adjacentes i e k de uma matriz binária B pode ser calculado pelo número de ocorrências do padrão 101 e 010 em cada linha r da matriz B,

dado pela Equação (3.3).

$$\delta(i,j,k) = \sum_{r=1}^{m} \left(B_j^r - B_i^r \times B_j^r - B_j^r \times B_k^r + B_i^r \times B_k^r \right) \tag{3.3}$$

Na implementação da ILS proposta, a δ-avaliação é utilizada por ambos os procedimentos de busca local, dado que ambos possuem uma vizinhança de tamanho $O(n^2)$ que é totalmente explorada.

3.3 Experimentos computacionais

Uma série de experimentos computacionais foram realizados com o intuito de definir os parâmetros do método proposto e aferir a qualidade das soluções reportadas. As seções a seguir descrevem esses experimentos e as instâncias utilizadas nos mesmos.

O ambiente computacional utilizado consiste de um computador com processador *Intel Core i7-8700* de 3.2 GHz com 16 GB de memória RAM sob o sistema operacional Ubuntu 18.04. A heurística construtiva e a metaheurística ILS foram implementadas utilizando-se a linguagem C++ e compilados utilizando-se *g++ 4.4.1* e a opção de otimização -O3.

As 45 instâncias artificiais e 5 instâncias reais utilizadas nos experimentos de Haddadi et al. (2015) são consideradas. Cada coluna das instâncias artificiais contém pelo menos um elemento não nulo e cada linha possui no mínimo dois blocos de uns. Conforme listado na Tabela 6, as instâncias artificiais são divididas em nove grupos de cinco instâncias cada. As instâncias reais, cujas dimensões são listadas na Tabela 7, surgem de um problema de localização de estações proposto por uma empresa ferroviária alemã.

Tabela 6 – Instâncias artificiais.

Grupo	A	B	C	D	E	F	G	H	I
Densidade (%)	2	5	10	2	5	10	2	5	10
Dimensões	100×200			100×500			100×1000		

Tabela 7 – Instâncias reais.

Instância	$RCOV_{1km}$	$RCOV_{2km}$	$RCOV_{3km}$	$RCOV_{5km}$	$RCOV_{10km}$
Dimensões	757×707	1196×889	1419×886	1123×593	275×165

3.3.1 Ajuste de parâmetros

Conforme mencionado na Seção 3.2.3, a implementação proposta para a ILS possui parâmetros relativos ao critério de parada e à porcentagem α de perturbação. Um critério de parada comum é a definição de um limite máximo de iterações *max_iter* para a ILS.

Entretanto, experimentos preliminares demonstraram que a utilização de um único valor estático não é o suficiente. Um valor pequeno de *max_iter*, embora possa ser adequado para instâncias pequenas e esparsas, pode levar a um tempo de execução proibitivo para instâncias grandes e densas. Assim, visando um melhor balanceamento entre o tempo de execução e a qualidade da solução, optou-se por considerar um limite dinâmico de tempo máximo de execução *max_time* como um segundo critério de parada.

Para a definição dos parâmetros utilizou-se o método *offline* de configuração automática de algoritmos de otimização *irace* (LÓPEZ-IBÁÑEZ et al., 2016). Dado um conjunto de instâncias do problema e um conjunto de possíveis valores para os parâmetros, o *irace* determina as melhores combinações de valores para os parâmetros. O conjunto de instâncias utilizado neste experimento é composto por 10% das instâncias utilizadas, selecionadas aleatoriamente. A Tabela 8 demonstra os valores considerados pelo *irace* na definição dos parâmetros e os valores selecionados.

Tabela 8 – Valores considerados e selecionados pelo *irace* para definição dos parâmetros.

Parâmetro	Opções	Valor selecionado
α (%)	{5, 10, 15, 20, 25, 30, 35, 40, 45, 50}	10
max_iter	{50, 100, 150, 200, 250}	150
max_time	$\{\frac{n}{3}, \frac{n}{2}, n, 2 \times n\}$	$\frac{n}{2}$

3.3.2 Comparação da ILS com o estado da arte

Após a definição dos parâmetros, os resultados obtidos pela ILS foram comparados com os resultados reportados por Haddadi et al. (2015). Infelizmente, uma comparação individual dos resultados não é possível, visto que Haddadi et al. (2015) reportam apenas a média dos valores das soluções. Para evitar tal inconveniente em eventuais trabalhos futuros, os resultados detalhados da ILS podem ser encontrados no material suplementar[1]. Além disso, não é possível fazer uma comparação justa entre os tempos de execução dos métodos pois foram executados em ambientes computacionais diferentes. Entretanto, a heurística proposta por Haddadi et al. (2015) é simples, sendo constituída de uma única aplicação de dois procedimentos de busca local, sem estratégias para escapar de eventuais ótimos locais não globais, devido a isso, seu tempo de execução é frequentemente desprezível.

As tabelas a seguir apresentam os resultados que constituem o atual estado da arte e a média obtida a partir de 20 execuções independentes da ILS para cada instância. Para a heurística de Haddadi et al. (2015), os valores médios de soluções reportados no trabalho original são apresentados (BKS). Para a ILS, as tabelas apresentam o valor da solução inicial obtida pela heurística construtiva (I), a melhor solução encontrada (S^*), a

[1] Disponível em <https://doi.org/10.1016/j.cor.2020.104948>

média dos valores de solução (S), o desvio padrão de S (σ), o tempo médio de execução em segundos (T) e a distância percentual (gap) entre a solução reportada pela ILS e o melhor valor de solução conhecido, calculado como $100 \times \frac{\text{Solução ILS - BKS}}{\text{BKS}}$. Note-se que a *Solução ILS* pode ser a melhor solução reportada S^* (gap^* (%)) ou a média dos valores de solução S (gap (%)). Os melhores valores de solução são destacados em negrito. A Tabela 9 apresenta os resultados para as 45 instâncias artificiais.

Tabela 9 – Resultados para as instâncias artificiais.

Grupo	Haddadi et al. (HADDADI et al., 2015)	ILS						
	BKS	*I*	S^*	gap^* (%)	S	gap (%)	σ	T
A	253,00	296,80	**221,40**	-12,49	222,93	-11,89	0,88	100,00
B	695,80	814,40	**634,20**	-8,85	636,74	-8,49	1,15	100,00
C	1358,60	1633,00	**1274,60**	-6,18	1278,02	-5,93	1,73	100,00
D	552,00	632,60	**475,40**	-13,88	478,45	-13,32	1,57	250,06
E	1616,00	1962,40	**1434,80**	-11,21	1442,39	-10,74	3,66	250,06
F	3308,40	4110,80	**3044,60**	-7,97	3057,30	-7,59	6,06	250,01
G	1072,40	1219,80	**902,20**	-15,87	908,84	-15,25	3,86	500,41
H	3125,40	3901,60	**2747,00**	-12,11	2757,92	-11,76	4,99	500,41
I	6375,40	8110,40	**5889,20**	-7,63	5904,16	-7,39	8,16	500,40

De acordo com os dados reportados, o método proposto apresenta melhores valores médios de solução para todos os grupos de instâncias artificias, superando os resultados do estado da arte. O *gap* médio reportado foi de $-10{,}26\%$, variando entre $-5{,}93\%$ e $-15{,}25\%$. Se considerarmos apenas o *gap* relativo as melhores soluções reportados pela ILS, o gap médio é de $-10{,}68\%$, variando entre $-6{,}18\%$ e $-15{,}87\%$. Os menores valores de *gap* são reportados para as maiores instâncias. O desvio padrão médio de apenas 3,56 demonstra a consistência da ILS em gerar soluções de alta qualidade com baixa variação em execuções independentes. O limite de tempo de execução foi o critério de parada predominante. Para as maiores instâncias (grupos G, H e I) o tempo de execução média foi de 8,3 minutos, um excelente valor considerando-se a qualidade das soluções e a dimensão das instâncias.

O tempo de execução da nova heurística construtiva foi inferior a 0,01 segundos para cada instância. As soluções reportadas apresentam um *gap* médio de 20,07% em relação as soluções reportadas por Haddadi et al. (2015), variando entre 13,75% e 27,22%. Em média, a ILS exigiu 10,45 iterações para melhorar as soluções iniciais em 35,92%.

Adicionalmente, uma segunda versão da ILS usando soluções iniciais geradas aleatoriamente (ILS_r) foi implementada para aferir de forma precisa sua convergência. Soluções aleatórias foram geradas em menos de 0,01 segundos para cada instância. O *gap* médio destas soluções, em comparação às soluções de Haddadi et al. (2015), foi de 54,59%, com valor máximo chegando a 95,50%. Em média, o ILS_r exigiu 8,09 iterações para melhorar a solução inicial em 61,49%. As soluções iniciais geradas pela heurística construtiva superam em qualidade as soluções iniciais geradas aleatoriamente com um *gap* médio de 29,39%, chegando ao máximo de 71,89%. Apesar da qualidade inferior das soluções iniciais geradas aleatoriamente, a ILS_r convergiu para soluções equivalentes às

reportadas por sua versão anterior sem aumentar o número de iterações, demonstrando a robustez do método proposto.

A Tabela 10, que segue o mesmo padrão definido para a tabela anterior, apresenta a média dos resultados para as cinco instâncias reais.

Tabela 10 – Resultados para as instâncias reais.

Instância	Haddadi et al. (HADDADI et al., 2015)	ILS						
	BKS	*I*	S^*	gap* (%)	*S*	gap (%)	σ	*T*
$RCOV_{1KM}$	**757,00**	757,00	**757,00**	0,00	**757,00**	0,00	0,00	281,64
$RCOV_{2KM}$	**1206,00**	1210,00	1210,00	0,33	1210,00	0,33	0,00	444,30
$RCOV_{3KM}$	1461,00	1449,00	**1449,00**	-0,82	**1449,00**	-0,82	0,00	444,29
$RCOV_{5KM}$	1143,00	1179,00	**1132,00**	-0,96	1139,45	-0,31	5,63	296,10
$RCOV_{10KM}$	280,00	287,00	**276,00**	-1,43	**276,00**	-1,43	0,00	82,00

Exceto pela instância $RCOV_{2KM}$ (gap de 0,33%), novos melhores valores de solução foram estabelecidos ou igualados para todas as instâncias. A única instância com uma solução ótima comprovada é $RCOV_{1KM}$, pois o valor de solução reportado para esta instância é igual a um limitante inferior trivial conhecido, o número de linhas da matriz binária. Entretanto, todos os valores de solução reportados para as instâncias deste conjunto aproximam-se deste limitante inferior, com um gap máximo reportado para instância $RCOV_{3KM}$ de apenas 2,11%.

A solução inicial gerada pela heurística construtiva coincide com o melhor valor de solução conhecido para três instâncias ($RCOV_{1KM}$, $RCOV_{2KM}$ e $RCOV_{3KM}$). Para as outras duas instâncias ($RCOV_{4KM}$ e $RCOV_{5KM}$), o gap entre a solução inicial e os valores de Haddadi et al. (2015) são de 3,15% e 2,5%, respectivamente. Em média, a ILS exige 7,85 iterações para melhorar a solução inicial em 3,47% e 3,99%, respectivamente, para as instâncias $RCOV_{4KM}$ e $RCOV_{5KM}$.

Uma análise estatística foi executada para comparar os resultados obtidos pela heurística construtiva e os reportados pelo método de Haddadi et al. (2015). O teste de normalidade *Shapiro-Wilk* (SHAPIRO; WILK, 1965), com um intervalo de confiança de 95%, confirmou a hipótese nula de que os resultados comparados, heurística construtiva ($W = 0,92275$, $p = 0,5478$) e o método de Haddadi et al. (2015) ($W = 0,94085$, $p = 0,6719$), poderiam ser modelados de acordo com uma distribuição normal. Assim, o *teste t* de *Student* (STUDENT, 1908) foi aplicado e indicou que não há diferença significativa entre os resultados comparados ($t = 0,88192$, $df = 4$, $p = 0,7862$) para um nível de significância de 0,05.

As execuções independentes da ILS resultaram em um desvio padrão de zero para quatro instâncias e de apenas 0,4% para a instância $RCOV_{5KM}$. Novamente, o tempo de execução aproxima-se de $\frac{n}{2}$, indicando que o critério de parada predominante é o limite de tempo. O tempo máximo de execução foi de 7,4 minutos para as maiores instâncias.

Análises estatísticas foram realizadas para comparar o método proposto com o

método de Haddadi et al. (2015). O teste de normalidade *Shapiro-Wilk* (SHAPIRO; WILK, 1965), com um intervalo de confiança de 95%, confirmou a hipótese nula de que os resultados comparados, ILS ($W = 0.75781$, $p = 0.001572$) e o método de Haddadi et al. (2015) ($W = 0.75315$, $p = 0.001391$), poderiam ser modelados de acordo com uma distribuição normal. O *teste t* de *Student* (STUDENT, 1908) foi aplicado e indicou que existe diferença significativa e que a ILS possui os melhores valores médios ($t = -3,0138$, $df = 13$, $p = 0,004986$) para um nível de significância de 0,05.

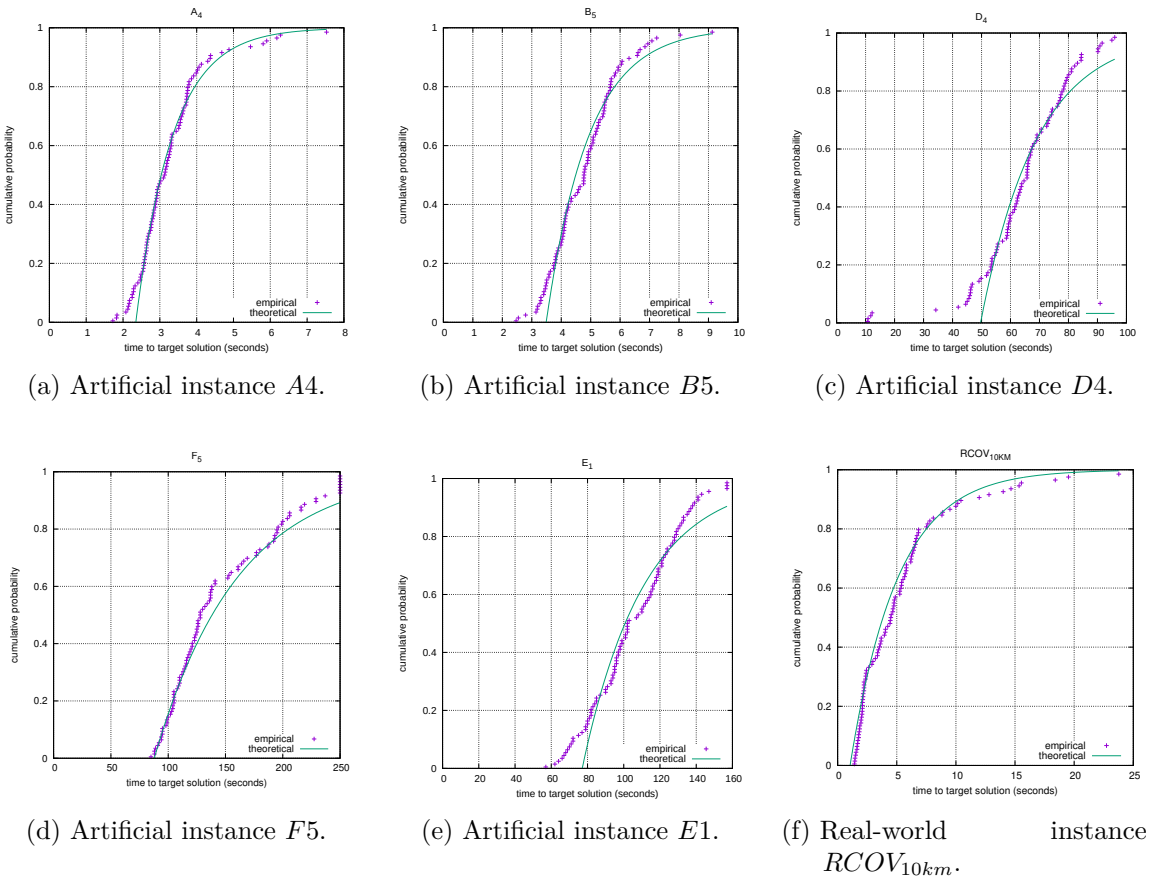
Embora a heurística de Haddadi et al. (2015) considere as características de permutação do CBM, a implementação adotada inevitavelmente converge para um ótimo local que não é necessariamente um ótimo global. Ao coordenar procedimentos de busca local eficientes com uma estratégia eficaz para escapar de ótimos locais, o método ILS proposto realiza uma melhor amostragem do espaço de solução e supera o atual estado da arte para o CBM.

Novamente, uma versão do ILS utilizando soluções iniciais geradas aleatoriamente foi implementada para aferir a convergência do ILS. O *gap* médio das soluções aleatórias, em relação às soluções de Haddadi et al. (2015) foi de 33,11%, variando entre 1,45% e 57,48%. Em média, a ILS_r exigiu 9,43 iterações para melhorar as soluções iniciais em 31,19%. Conforme relatado anteriormente, apesar da qualidade inferior das soluções iniciais aleatórias, a ILS_r convergiu para soluções com a mesma qualidade de sua versão original sem um aumento no número de iterações necessárias. O teste de normalidade *Shapiro-Wilk* (SHAPIRO; WILK, 1965), com intervalo de confiança de 95%, confirmou a hipótese nula de que os resultados comparados, ILS ($W = 0,79402$, $p = 0,0000176$) e ILS_r ($W = 0,79423$, $p = 0,0000178$), poderiam ser modelados de acordo com uma distribuição normal. O *teste t* de *Student* (STUDENT, 1908) foi aplicado e indicou que não há uma diferença significativa entre os resultados comparados ($t = 0,33277$, $df = 44$, $p = 0,6296$) para um nível de significância de 0,05.

Gráficos do tipo *time-to-target* (AIEX; RESENDE; RIBEIRO, 2007), ou gráficos ttt, são utilizados para ilustrar a convergência da ILS. A hipótese por trás dos gráficos ttt é que o tempo de execução de um método se ajusta a uma distribuição exponencial se o método for executado uma quantidade suficiente de vezes. Seis instâncias foram aleatoriamente selecionadas para gerar gráficos ttt. Para cada instância a ILS foi executada independentemente 100 vezes e reportou o número de iterações necessárias para atingir um valor de solução alvo no máximo 5% maior que a melhor solução conhecida. Em seguida, comparou-se a distribuição resultante (*empirical*, representada pelas cruzes roxas) com a distribuição exponencial (*theoretical*, representada pela linha verde). Os gráficos ttt apresentam a probabilidade cumulativa (eixo y) de o método atingir uma solução alvo em cada número de iterações (eixo x).

A Figura 7 (a) - (f) ilustra a convergência da ILS para seis instâncias. Todos os

gráficos são interpretados de forma análoga. Por exemplo, para a instância artificial A_4, ilustrada em (a), é possível observar que a probabilidade da ILS reportar uma solução tão boa quanto o valor alvo em 7 segundos é de 99%, confirmando a eficiência do segundo critério de parada estabelecido, o limite dinâmico do tempo de execução.



(a) Artificial instance $A4$. (b) Artificial instance $B5$. (c) Artificial instance $D4$.

(d) Artificial instance $F5$. (e) Artificial instance $E1$. (f) Real-world instance $RCOV_{10km}$.

Figura 7 – Gráficos ttt ilustrativos para 6 instâncias.

3.4 Conclusão

Neste capítulo abordou-se o CBM, um problema $\mathcal{NP}$-Difícil com ampla aplicação teórica e prática em diversos contextos. Foram apresentadas uma nova representação para o problema utilizando-se grafos, uma heurística construtiva e uma implementação da metaheurística ILS para resolução do CBM. A qualidade dos métodos propostos foi avaliada considerando-se o único conjunto *benchmark* de instâncias disponível na literatura e o método estado da arte para o CBM. Os experimentos computacionais realizados demonstraram que a nova representação em grafos e a heurística construtiva são eficazes na geração de boas soluções iniciais em baixo tempo computacional. A implementação proposta da metaheurística ILS mostrou-se eficaz na produção de soluções de alta qualidade, estabelecendo novos melhores resultados para todos os nove grupos de instâncias artificiais e para quatro de cinco instâncias do conjunto de instâncias reais. Considerando-se os

resultados reportados e as análises estatísticas realizadas, o método proposto estabelece novos valores de referência para o conjunto *benchmark* de instâncias disponível e torna-se o novo estado da arte para o CBM.

4 Sequenciamento de tarefas em máquinas paralelas com limitação de recursos

Este capítulo é fortemente baseado no artigo *"Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling in a flexible manufacturing system"* de autoria de Leonardo Cabral da Rocha Soares e Marco Antonio Moreira de Carvalho. Submetido ao periódico *Computers & Operations Research*, o artigo encontra-se em processo de revisão. O artigo aborda o problema de sequenciamento de tarefas em máquinas paralelas com limitação de recursos utilizando a metaheurística paralela algoritmo genético de chaves aleatórias viciadas e propõe um novo conjunto *benchmark* de instâncias para o problema. A seguir, são apresentados os fundamentos teóricos do problema abordado, os trabalhos relacionados, os métodos computacionais utilizados, os experimentos realizados e as conclusões.

O processamento de tarefas em SMFs demanda a disponibilidade de uma série de recursos, tais como máquinas flexíveis, ferramentas, paletes, manipuladores automáticos de matérias-primas, entre outros. Quando tais recursos geram limitações ao processamento de tarefas em um ambiente composto por máquinas flexíveis em paralelo, temos o *Problema de Sequenciamento em Máquinas Paralelas com Limitações de Recursos* (*Resource Constrained Parallel Machine Scheduling*, RCPMS). Primeiramente estudado por Garey e Graham (1975), o RCPMS pertence à classe de problemas $\mathcal{NP}$-Difícil, visto que, se considerarmos cada máquina do RCPMS isoladamente, temos uma instância do problema de sequenciamento de tarefas em uma máquina, um problema provado pertencer à classe $\mathcal{NP}$-Difícil por Crama et al. (1994).

Entre os diversos exemplos de possíveis limitações de recursos em SMFs, podemos citar a indisponibilidade de matéria-prima em algum momento da produção (PINHEIRO; ARROYO, 2020), a indisponibilidade de máquinas devido a características intrínsecas às tarefas que demandem o processamento em máquinas específicas (LI et al., 2020) ou devido a paradas para manutenção (REDDY et al., 2018) e o compartilhamento de ferramentas por máquinas flexíveis (KEUNG; IP; LEE, 2001), incluindo-se ambientes em que as máquinas flexíveis não possuem *magazine* (AGNETIS et al., 1997). Essas limitações podem ocorrer em conjunto ou isoladamente.

Um exemplo de aplicação prática do RCPMS pode ser encontrado nas indústrias de fabricação de componentes microeletrônicos (CHUNG et al., 2019), em especial na fabricação de microchips. Centenas de circuitos são impressos em um *wafer* semicondutor por um processo denominado fotolitografia. O desenho de cada circuito é criado em uma ferramenta denominada máscara. Esta máscara é utilizada como um molde durante o

processo de impressão dos circuitos. Conforme ilustrado pela Figura 8, uma luz atravessa a máscara, formando o padrão do circuito que será impresso. Através de lentes, o tamanho do padrão é reduzido e gravado sobre o *wafer* já revestido com material fotossensível. O mesmo padrão de circuito pode ser impresso no *wafer* centenas de vezes.

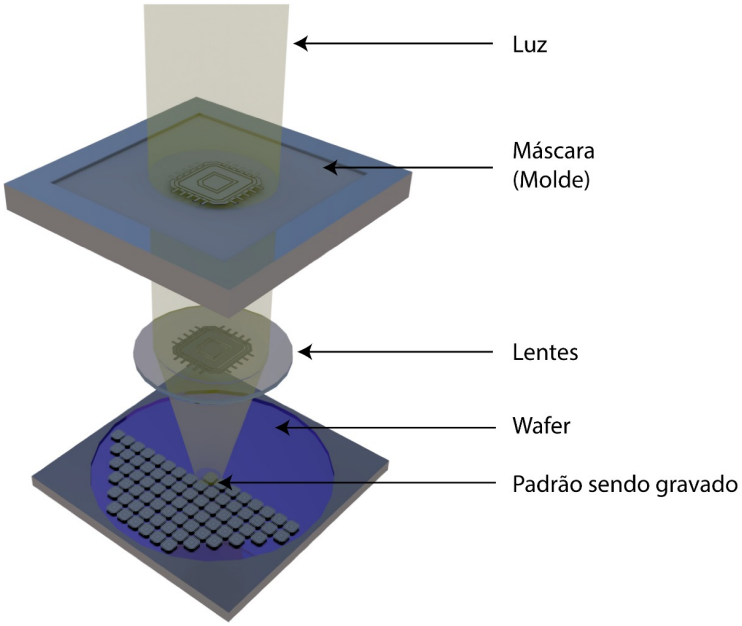


Figura 8 – Processo de gravação de padrão de circuito em *wafer*.

As máquinas e moldes utilizados na fabricação dos *wafers* possuem custo elevado (CHUNG et al., 2019). Considerando que toda máquina possui capacidade para produzir qualquer modelo de microchip, é comum que o número de cópias de cada molde seja inferior ao número de máquinas, implicando em compartilhamento de moldes. Em ambientes de produção em larga escala com máquinas paralelas, a utilização dos moldes deve ser planejada de forma que a produção seja otimizada, minimizando-se esperas pela liberação de moldes.

Tal aplicação prática do RCPMS é uma especificação do IPMTC, descrito no Capítulo 2. As ferramentas correspondem aos moldes, o processamento de tarefas corresponde a impressão dos circuitos no *wafer* e o tempo necessário para a troca de uma ferramenta corresponde ao tempo de configuração necessário para a montagem de um molde na máquina. O *magazine* das máquinas possui capacidade igual a um e o conjunto total de ferramentas é compartilhado por todas as máquinas. Cada tarefa utiliza apenas uma ferramenta e, exceto em caso de menções explícitas em contrário, há apenas uma cópia de cada ferramenta.

A formalização desta aplicação prática do RCPMS segue o mesmo padrão estabelecido para o IPMTC. Dada a restrição originada pelo compartilhamento de moldes, duas tarefas que utilizem o mesmo molde não podem ser processadas no mesmo intervalo

de tempo em máquinas diferentes. De acordo com a notação fornecida por Graham et al. (1979), esta aplicação do RCPMS é classificada como $P_m|res1|C_{max}$, em que P_m indica o ambiente industrial composto por máquinas paralelas idênticas, $res1$ refere-se a restrição originada pelo compartilhamento de moldes e C_{max} a função objetivo de minimização do *makespan*.

Uma instância válida para esta aplicação consiste na quantidade de máquinas que compõem o ambiente de produção, uma lista de tarefas com seus respectivos tempos de processamento, o molde utilizado por cada tarefa e o tempo de configuração necessário para a montagem de cada molde. A Tabela 11 apresenta um modelo de instância válida em que se considera $m = 2$ e $\bar{p} = 1$.

Tabela 11 – Exemplo de instância do RCPMS.

Tarefas	Tempo de processamento	Moldes
1	10	1
2	4	4
3	3	2
4	1	1
5	6	3
6	5	2
7	8	3
8	4	4
9	9	1
10	3	2

A solução deste problema é composta pela lista de tarefas atribuídas e sequenciadas em cada máquina e o valor de *makespan* obtido. Para uma solução ser considerada viável, a restrição referente a utilização de moldes deve ser respeitada. A Figura 9 apresenta uma solução inviável para a instância exemplificada. As tarefas são representadas por retângulos preenchidos com larguras proporcionais ao tempo de processamento das mesmas e identificadas de acordo com o número associado a elas. Os moldes utilizados são representados pelas cores dos retângulos, conforme detalhado na legenda da figura. O tempo de configuração para a troca dos moldes é representado pelo retângulo hachurado.

No exemplo mostrado na Figura 9, a utilização do molde 1 por duas máquinas no intervalo de tempo contido entre 0 e 9 viola a restrição. Para que a solução seja considerada viável, o conflito gerado pela utilização simultânea do molde 1 precisa ser resolvido. Isso pode ser feito acrescentando-se a uma das máquinas um período ocioso de espera pela liberação do molde em uso ou por uma nova alocação de tarefas em intervalos de tempo que elimine o conflito.

A Figura 10, que segue o mesmo padrão descrito para a figura anterior, demonstra uma solução viável para a instância apresentada na Tabela 11. Neste exemplo, foram alocadas à máquina 1 as tarefas [1, 4, 5, 2, 10, 3] nesta ordem. O tempo de processamento

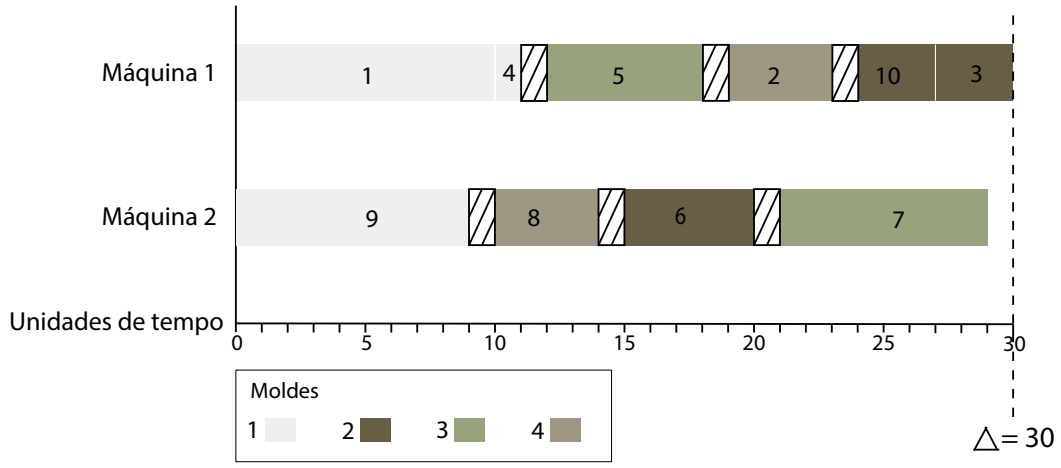


Figura 9 – Exemplo de solução inviável para a instância apresentada na Tabela 11.

da máquina 1 é obtido pela soma do tempo de processamento das tarefas a ela atribuídas com o número de troca de moldes necessários multiplicado pelo tempo de configuração utilizado para cada troca, ou seja, $10 + 1 + 6 + 4 + 3 + 3 + (3 \times 1) = 30$ unidades de tempo. Para a máquina 2 foram alocadas as tarefas [7, 8, 6, 9], resultando em um tempo de processamento igual a 29 unidades de tempo. O *makespan* da solução é igual ao maior tempo de processamento entre as máquinas que a constituem, neste caso, a máquina 1 com 30 unidades de tempo.

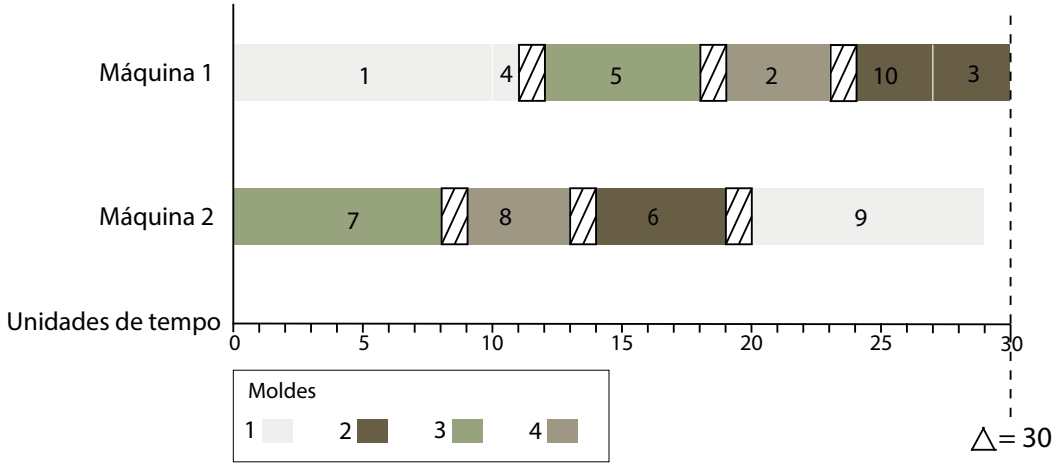


Figura 10 – Exemplo de solução viável para a instância apresentada na Tabela 11.

A função objetivo do RCPMS para a aplicação demonstrada é a minimização do *makespan*, resultado da Equação (4.1). Sejam a variável I_j o tempo ocioso gerado imediatamente antes do processamento da tarefa j e a variável $x_{ij} = 1$ caso a tarefa j seja processada pela máquina i ou $x_{ij} = 0$ caso contrário. A equação realiza a soma entre o tempo de processamento das tarefas alocadas à máquina, o tempo ocioso gerado pelo

escalonamento destas destas tarefas e a multiplicação entre o número de trocas de moldes necessários e o tempo de configuração exigido para a troca de um molde. O número de troca de moldes é determinado pela Equação (2.2) proposta originalmente para o SSP.

$$\Delta = \max\left\{ \sum_{j \in J} x_{ij} p_j + \sum_{j \in J} x_{ij} I_j + Z_{SSP}(R^i) \times \overline{p} \right\}, \ \forall i \in M \tag{4.1}$$

Apenas configurações realizadas antes do início do tempo de processamento das máquinas, $t \leq 0$, não são consideradas. Assim, caso o processamento de uma máquina se inicie com um período ocioso, o tempo de configuração para o carregamento do primeiro molde será contabilizado.

4.1 Trabalhos relacionados

Estudos relacionados ao RCPMS estão presentes na literatura desde a década de 1970. Os primeiros autores a abordarem o tema foram Garey e Graham (1975). É interessante ressaltar que os autores descrevem a inclusão da limitação de recursos no modelo proposto como um item para fornecer maior realismo ao trabalho. Ao longo dos anos, problemas de escalonamento com restrições de recursos foram abordados em diversas áreas, tais como enfermagem (OHKI; MORIMOTO; MIYAKE, 2006), competições esportivas (GUANGDONG; PING; QUN, 2007), indústrias alimentícias (KARRAY; BENREJEB; BORNE, 2011), entre outras (PINEDO, 2008).

Considerando-se o RCPMS especificamente em contextos industriais de manufatura flexível compostos por máquinas paralelas, salvo melhor juízo, a primeira publicação ocorreu mais de uma década após o trabalho seminal da versão geral do problema. Apesar desta lacuna inicial, diversos trabalhos são encontrados na literatura abordando diferentes cenários e tipos de recursos. Nos parágrafos a seguir, alguns destes cenários são brevemente introduzidos e os principais trabalhos relacionados são listados. Particularmente, será dada maior ênfase aos trabalhos que utilizam moldes como recursos compartilhados, sendo apresentados em ordem cronológica.

O compartilhamento de ferramentas por máquinas flexíveis idênticas paralelas aparece na literatura como a primeira aplicação prática do RCPMS em SMFs. Neste ambiente, durante o processamento sequencial de tarefas, pode ser necessário efetuar trocas de ferramentas no *magazine*. Caso uma ferramenta necessária ao processamento de uma tarefa esteja em uso por outra máquina, é necessário esperar até que o processamento termine para que a ferramenta possa ser movida do *magazine* de uma máquina para o outro. Quando não estão em uso, as ferramentas permanecem em um depósito centralizado e podem ser carregadas no *magazine* de qualquer máquina sem tempo de espera. Neste cenário, as ferramentas são o recurso limitado compartilhado que caracteriza o RCPMS.

Com pequenas variações e diferentes objetivos, este cenário foi considerado em diversos trabalhos da literatura (HAN; NA; HOGG, 1989; AGNETIS; NICOLO; LUCERTINI, 1991; ROH; KIM, 1997; ÖZPEYNIRCI, 2015).

Em alguns ambientes industriais, o processamento de tarefas por máquinas paralelas demanda algumas operações manuais. Os trabalhadores destas indústrias são aptos a operar diferentes máquinas, e devido a esta interação a velocidade de processamento é diretamente afetada pelo número de trabalhadores alocados a cada máquina. Desta forma, os trabalhadores são considerados um recurso limitado compartilhado entre as máquinas. A referência à mão de obra como um recurso limitado compartilhado é recorrente na literatura, embora não necessariamente no contexto de manufatura flexível (DANIELS; HOOPES; MAZZOLA, 1997; DANIELS; HUA; WEBSTER, 1999; RUIZ-TORRES; LÓPEZ; HO, 2007; RUIZ-TORRES; CENTENO, 2007; EDIS; ARAZ; OZKARAHAN, 2008; EDIS; OZKARAHAN, 2012).

A literatura específica sobre o RCPMS com limitações originadas pela utilização de moldes, é restrita. A maioria dos trabalhos são estudos de caso isolados. Nestes estudos, os resultados gerados são comparados com o escalonamento manual ou com outros métodos propostos no mesmo trabalho. As principais aplicações práticas envolvem a moldagem de produtos plásticos, metálicos e a produção de componentes eletrônicos.

Tamaki et al. (1993) abordam o escalonamento de tarefas em máquinas paralelas não-relacionadas com restrições de moldes. O problema tem sua origem em fábricas de tubos de plástico. Para ser fabricado, cada tubo utiliza um determinado tipo de molde. Os moldes não são universais, sendo necessária a compatibilidade entre máquina e molde. A quantidade de cada molde é limitada e são considerados tempos de configuração para montagem e desmontagem dos moldes. Além de uma formulação matemática para o problema, os autores apresentam uma implementação da metaheurística *simulated annealing*, um algoritmo genético e dois métodos baseados em buscas locais. Três objetivos foram considerados nos experimentos realizados: o tempo máximo de conclusão, o atraso médio e o atraso máximo. Os métodos implementados foram comparados entre si, utilizando-se duas instâncias propostas no próprio artigo. Os resultados reportados pela metaheurística *simulated annealing* superaram as demais implementações.

Gao et al. (1998) abordam o RCPMS para o gerenciamento de produção em uma fábrica de eletrodomésticos. Nesta fábrica, componentes plásticos para eletrodomésticos são produzidos por máquinas paralelas. Cada componente possui um tempo de processamento e utiliza um determinado molde. Há uma relação de compatibilidade entre máquinas e moldes e o número de cópias de cada molde é limitado. Somente o tempo de configuração para montagem do molde é considerado. São apresentados um modelo matemático e uma heurística baseada em processamento de listas. As abordagens foram avaliadas utilizando-se uma instância real e comparado ao escalonamento manual. Os resultados mostraram uma

redução média de 68,00% no tempo de processamento diário.

O mesmo cenário industrial foi posteriormente abordado por Xianzhang e Chengyao (1999). Visando a minimização do total de tarefas atrasadas, os autores apresentam uma heurística baseada no processamento de listas. Os experimentos foram realizados utilizando-se uma instância real de grande proporção. Os resultados obtidos foram comparados com o escalonamento manual e mostraram uma redução de 93,31% no total de tarefas atrasadas.

A produção de tubos de plástico é retomada no trabalho de Chen (2005), com o objetivo de minimizar o *makespan*. É apresentada uma implementação da metaheurística busca tabu para o problema e os experimentos utilizaram 145 instâncias propostas no próprio artigo. Os resultados reportados foram comparados com uma implementação da metaheurística *simulated annealing* e com resultados ótimos obtidos a partir de um modelo matemático, ambos propostos por Tamaki et al. (1993). A busca tabu apresenta melhores resultados, um menor tempo de processamento e foi capaz de alcançar os resultados ótimos para 24 instâncias pequenas.

Considerando o mesmo ambiente, mas com o objetivo de minimizar o atraso total, Chen e Wu (2006) apresentam uma heurística híbrida, combinando o método *threshold accepting* e busca tabu. Os experimentos utilizaram dois conjuntos de instâncias propostos no próprio artigo. Para o primeiro conjunto, contendo os menores problemas, o método proposto reportou soluções ótimas para todas as instâncias, embora o método utilizado para gerar os resultados ótimos não tenha sido mencionado. Para o segundo conjunto, contendo instâncias grandes, os valores obtidos foram comparados com outros três outros métodos: o *simulated annealing* proposto por Tamaki et al. (1993), uma versão não hibridizada do método *threshold accepting* e uma nova adaptação do método *apparent tardiness cost with setups*, extendido por Lee e Pinedo (1997) para o escalonamento de tarefas em máquinas paralelas idênticas com tempo de preparo dependente da sequência. A heurística híbrida proposta reporta os melhores valores de solução entre os métodos comparados.

Em um ambiente composto por máquinas paralelas não-relacionadas, Hong, Sun e Li (2008) abordam o escalonamento de tarefas que podem ser processadas utilizando-se um molde de um subconjunto de moldes disponíveis. Cada molde pode ser montado em um subconjunto de máquinas compatíveis, e a velocidade de processamento das tarefas varia de acordo com a máquina. O problema é modelado matematicamente e resolvido com uma heurística baseada em processamento de listas. Os experimentos utilizaram dados reais de uma indústria de tubos de aço. Foram comparados o atraso total, o número de tarefas atrasadas e o atraso médio obtidos pela heurística com os valores obtidos pelo escalonamento manual. Embora não reportem valores específicos, os autores relatam que a abordagem proposta melhora significativamente os resultados.

O escalonamento de tarefas com restrições de moldes em máquinas paralelas idênticas foi abordado primeiramente por Hong, Jou e Sun (2009). No ambiente tratado,

há apenas uma cópia de cada molde e o tempo de configuração necessário para montagem de qualquer molde é igual a três. Com o objetivo de minimizar o *makespan* são propostas duas versões de um algoritmo genético. Enquanto a primeira versão simplesmente descarta soluções inviáveis, a segunda emprega um operador de ajuste que elimina os conflitos pela realocação das tarefas conflitantes. Os experimentos realizados compararam as duas versões de algoritmo genético e mostraram que o operador de ajuste efetivamente melhora as soluções reportadas.

O trabalho apresentado por Xiong, Fan e Li (2013), considera o escalonamento de lotes com restrições de moldes. Cada lote é formado por um conjunto de peças que devem ser moldadas e pintadas. Todas as peças de um mesmo lote utilizam o mesmo subconjunto de moldes e devem ser pintadas da mesma cor. Cada molde pode ser montado em um subconjunto de máquinas e a velocidade de processamento de cada máquina varia conforme o molde montado na mesma. Além do tempo de configuração para montagem e desmontagem dos moldes, a troca de cores entre o processamento sequencial de lotes também pode demandar tempo de configuração. Visando minimizar o atraso total ponderado, os autores apresentam um algoritmo genético e duas heurísticas baseadas em processamento de listas. Apenas uma instância artificial foi utilizada nos experimentos realizados. Os resultados reportados apontaram superioridade do algoritmo genético em comparação às heurísticas propostas.

Lee et al. (2014) apresentam um algoritmo genético hibridizado com buscas locais para minimizar o *makespan* em um ambiente composto por máquinas paralelas não-relacionadas. Para ser processada, cada tarefa necessita de um molde, dentro de um subconjunto de moldes disponíveis. Cada molde pode ser montado em um subconjunto de máquinas compatíveis. Os experimentos consideraram nove instâncias artificiais propostas no próprio artigo. Foram comparados o algoritmo genético e uma segunda versão de mesmo sem as buscas locais. Embora os resultados reportados não sejam detalhados, os autores relatam que a incorporação das buscas locais no algoritmo genético contribui efetivamente para melhoria da qualidade das soluções obtidas.

Após alguns anos sem publicações diretamente relacionadas, Chung et al. (2019) retomam o tema considerando uma indústria de componentes eletrônicos. O cenário abordado é o de impressão de *wafers* detalhado no início deste capítulo. Duas máquinas paralelas idênticas são empregadas, há apenas uma cópia de cada molde e o tempo de configuração para montagem dos moldes nas máquinas não é considerado.

Com o objetivo de minimizar o *makespan*, são apresentados um modelo matemático e duas heurísticas baseadas em processamento de listas. A primeira heurística agrupa tarefas por moldes em comum e a segunda ordena as tarefas por tempo de processamento. As aplicações destas duas heurísticas também foram combinadas entre si, dando origem a um terceiro método. Para fins de avaliação das heurísticas propostas, os autores adicionalmente

apresentaram a implementação das metaheurísticas *Discrete Particle Swarm Optimal* (DPSO) e *Variable Neighborhood Search* (VNS). Entretanto, ambas metaheurísticas são implementadas desconsiderando-se qualquer característica intrínseca ao problema. Os resultados gerados foram também avaliados em relação ao um limitante inferior dado pela solução ótima de um modelo relaxado, desconsiderando-se as restrições de moldes.

Os experimentos consideraram dois conjuntos de instâncias propostos no próprio artigo com o número de tarefas variando entre 6 e 150, número de máquinas igual a 2 e tempo de configuração igual a 0. Para o primeiro conjunto de instâncias, contendo os menores problemas, o modelo proposto apresentou resultados ótimos para todas as instâncias. As implementações das metaheurísticas VNS e DPSO reportaram um *gap* médio de 0,002% e 0,010% respectivamente. A aplicação composta das heurísticas de processamento de listas reportou um gap médio de 1,270%. Para o segundo conjunto de instâncias, considerando-se o limite de tempo estabelecido em 1200 segundos, o modelo proposto mostrou-se ineficaz. Assim, os autores utilizaram uma simplificação do modelo proposto por Mokotoff (2004) para obter limitantes inferiores. A simplificação do modelo utilizado é uma relaxação do problema tratado por não considerar os recursos compartilhados. Em comparação aos limites inferiores reportados, a aplicação composta das heurísticas alcança os melhores resultados, reportando um *gap* médio de 0,18%. As implementações das metaheurísticas VNS e DPSO reportaram, respectivamente, *gaps* médios de 0,98% e 1,42%.

Especificamente para o RCPMS no contexto de restrições originadas do compartilhamento de moldes, o trabalho de Chung et al. (2019) apresenta o único conjunto de instâncias *benchmark* disponível na literatura. Entretanto, a limitação do número de máquinas a apenas duas, o fato de não serem considerados os tempos de configuração necessários para a troca dos moldes e os baixos valores médios de *gap* reportados em comparação com a solução da relaxação do problema, indicam que tal conjunto é constituído em sua maior parte por instâncias triviais, nas quais o uso de moldes não possui impacto substancial na dificuldade de solução. Desta maneira, surge a necessidade de criação de novas instâncias, que representem um desafio maior aos métodos propostos, possibilitando assim a progressão do estado da arte relacionado ao problema. Nos experimentos computacionais relatados na Seção 4.3 este conjunto *benchmark* de instâncias é analisado de forma detalhada e é considerado, juntamente com os os resultados disponíveis na literatura, para aferição da qualidade do método proposto.

4.2 Métodos

Esta seção descreve os métodos da literatura utilizados para comparações, bem como o método proposto para abordagem do problema. São descritos (i) a formulação matemática geral para o escalonamento de tarefas em máquinas (PINEDO, 2008), utilizado

como limitante inferior; (ii) uma formulação matemática para o escalonamento de tarefas em máquinas flexíveis com restrições de moldes (CHUNG et al., 2019); (iii) duas heurísticas baseadas em processamento de listas (CHUNG et al., 2019); e (iv) um algoritmo genético de chaves aleatórias viciadas hibridizado com quatro buscas locais.

4.2.1 Formulação matemática geral para o escalonamento de tarefas em máquinas

A formulação matemática abaixo oferece um modelo geral para o escalonamento de tarefas em máquinas idênticas paralelas (PINEDO, 2008). O modelo utiliza um conjunto de variáveis binárias x_{ij} igual a 1 se, e somente se, a tarefa j for processada pela máquina i. Sendo p_j o tempo de processamento da tarefa j, o modelo é dado por:

$$\text{Minimizar} \quad \Delta \tag{4.2}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad i \in M \tag{4.3}$$

$$\sum_{i \in M} x_{ij} p_i \leq \Delta \qquad j = 1,\ldots,n \tag{4.4}$$

$$x_{ij} \in \{0,1\} \qquad i \in M, j = 1,\ldots,n \tag{4.5}$$

A função objetivo (4.2) minimiza o *makespan* Δ. A restrição (4.3) obriga a alocação de todas as tarefas e garante que cada tarefa seja alocada em apenas uma máquina. A restrição (4.4) calcula o tempo de processamento total de cada máquina do conjunto M e define o *makespan*. A restrição (4.5) define a integralidade das variáveis de decisão. Este modelo foi utilizado na geração de limitantes inferiores para todas as instâncias utilizadas nos experimentos computacionais descritos na Seção 4.3.

4.2.2 Formulação matemática para o escalonamento de tarefas em máquinas flexíveis com restrições de moldes

Chung et al. (2019) apresentam um modelo para o escalonamento de tarefas em duas máquinas flexíveis paralelas considerando restrições originadas pelo compartilhamento de moldes e desconsiderando o tempo de configuração necessário para a troca de moldes. O modelo utiliza quatro famílias de variáveis binárias:

- y_j, igual a 1 se, e somente se, a tarefa j é a primeira a ser processada em um das máquinas;

- x_{kj}, igual a 1 se, e somente se, a tarefa k for escalonada imediatamente antes da tarefa j na mesma máquina;
- $x_{k,n+1}$, igual a 1 se, e somente se, a tarefa k for a última tarefa a ser processada em uma das máquinas;
- b_{kj}, igual a 1 se, e somente se, a tarefa k iniciar seu processamento antes do início do processamento da tarefa j.

Sabendo-se que os moldes são representados por diferentes inteiros positivos t_k, o molde utilizado por duas tarefas é o mesmo se a diferença entre eles for igual a zero. Assim, define-se $z_{kj} = (t_k - t_j)^2$. Sendo μ um inteiro suficientemente grande e c_j o tempo de conclusão da tarefa j, o modelo é dado por:

$$\text{Minimizar} \quad \Delta \tag{4.6}$$

$$\text{Sujeito a} \quad \sum_{j=1}^{n} y_j \leq 2 \tag{4.7}$$

$$y_j + \sum_{k=1,k\neq h}^{n} x_{kj} = 1 \qquad j = 1,\ldots,n \tag{4.8}$$

$$\sum_{k=1,k\neq h}^{n+1} x_{jk} = 1 \qquad j = 1,\ldots,n \tag{4.9}$$

$$c_j - p_j y_j \geq 0 \qquad j = 1,\ldots,n \tag{4.10}$$

$$b_{jk} + b_{kj} = 1 \qquad k,j = 1,\ldots,n, k \neq h \tag{4.11}$$

$$\sum_{k=1,k\neq h}^{n} (b_{jk} + b_{kj}) + 1 = n \qquad j = 1,\ldots,n-1 \tag{4.12}$$

$$c_j - c_k + \mu(1 - b_{kj}) + \mu z_{kj} \geq p_j \qquad k,j = 1,\ldots,n, k \neq h \tag{4.13}$$

$$c_j - c_k + \mu(1 - x_{kj}) \geq p_j \qquad k,j = 1,\ldots,n, k \neq h \tag{4.14}$$

$$\sum_{k=1}^{n} x_{k,n+1} \leq 2 \tag{4.15}$$

$$x_{kj} + x_{jk} \leq 1 \qquad k,j = 1,\ldots,n, k \neq h \tag{4.16}$$

$$\Delta - c_j \geq 0 \qquad j = 1,\ldots,n \tag{4.17}$$

$$b_{kj}, x_{kj}, y_j \in 0,1 \tag{4.18}$$

$$\Delta, c_k \geq 0 \tag{4.19}$$

A função objetivo (4.6) minimiza o *makespan* Δ. A restrição (4.7) garante que no máximo duas tarefas sejam as primeiras a serem processadas, uma em cada em máquina. A

restrição (4.8) define que uma tarefa deve ser a primeira a ser processada ou ser precedida por outra tarefa. De forma similar, a restrição (4.9) garante que uma tarefa seja sucedida por outra tarefa ou seja a última a ser processada em uma das máquinas. A restrição (4.10) garante que o tempo de conclusão da primeira tarefa, em cada máquina, seja maior ou igual ao seu tempo de processamento. As restrições (4.11) e (4.12) garantem que a ordem de processamento das tarefas seja determinada pelas variáveis de decisão e que o total de tarefas que precedem e sucedem uma tarefa j seja igual a $n-1$. A restrição (4.13) especifica que o tempo de conclusão de uma tarefa j deve ser maior ou igual a soma dos tempos de processamento das tarefas k e j se, a tarefa k precede a tarefa j e ambas utilizam o mesmo molde. A restrição (4.14) garante que o tempo de conclusão da tarefa j é maior ou igual a soma do seu tempo de processamento e do tempo de conclusão da tarefa que a precede. A restrição (4.15) garante que no máximo duas tarefas sejam as últimas a serem processadas, uma em cada máquina. A restrição (4.16) determina que a sequência de tarefas em cada máquina é dada pelas variáveis de decisão. O *makespan* é definido pela restrição (4.17). Por fim, as restrições (4.18) e (4.19) definem o domínio das variáveis de decisão.

Ao optar por não utilizar indexação no tempo para o modelo proposto, os autores apresentam um modelo compacto, com menos variáveis. Entretanto, devido a tal característica, o modelo não fornece uma solução que explicite a ordem de processamento das tarefas, apresentando somente a ordem relativa de cada tarefa em relação as demais. Para criação da ordem de processamento das tarefas em cada máquina, é necessário realizar um pós processamento da resposta do modelo.

Visando diminuir o tempo de processamento para provar a otimalidade das soluções, o valor da solução gerado pelo modelo de escalonamento geral descrito na Seção 4.2.1 foi adicionado a este modelo como limitante inferior. Da mesma forma, este limitante inferior foi utilizado como um dos critérios de parada do algoritmo genético de chaves aleatórias viciadas descrito na Seção 4.2.4.

4.2.3 Heurísticas baseadas em processamento de listas

Diferentes técnicas de processamento de listas são utilizadas na alocação de tarefas à máquinas flexíveis, diferindo entre si pelas regras empregadas para selecionar as tarefas e para selecionar as máquinas em uma atribuição tarefa × máquinas. Graham (1969) apresenta a regra tempo de processamento mais longo (*longest processing time*, LPT) para seleção de tarefas. Nela, as tarefas devem ser ordenadas em uma lista de forma não crescente pelo tempo de processamento. A partir da lista, as tarefas são atribuídas às máquinas de acordo com uma segunda regra, por exemplo, a máquina que se tornar ociosa primeiro.

Baseado na regra LPT, Chung et al. (2019) apresentam duas heurísticas para o

escalonamento de tarefas em máquinas paralelas com restrições de moldes. A primeira heurística, denominada pelos autores como *two-stage* LPT (TLPT), agrupa as tarefas gerando subconjuntos em que todas as tarefas requerem o mesmo molde. Os subconjuntos são ordenados de acordo com o somatório do tempo de processamento de todas as tarefas pertencentes ao mesmos. Com exceção do subconjunto com o maior tempo de processamento, os subconjuntos são alocados às máquinas de acordo com a regra LPT. As tarefas do subconjunto não alocado são consideradas individualmente de acordo com a regra LPT. Visando resolver conflitos resultantes do compartilhamento de moldes, as tarefas deste último subconjunto alocadas à uma das máquinas são movidas para o início do escalonamento. Dada a necessidade de ordenamento dos subconjuntos de tarefas, a complexidade desta heurística é limitada por $\mathcal{O}(n\ log\ n)$.

A segunda heurística proposta, denominada *Different* LPT (DMLPT), ordena as tarefas de acordo com a regra LPT. Entretanto, na fase de alocação das tarefas às máquinas, é selecionada a próxima tarefa disponível cujo molde requerido seja diferente do utilizado pela tarefa alocada anteriormente. Caso não haja uma tarefa que atenda a este critério, todas as tarefas restantes são alocadas à mesma máquina. A etapa de seleção da próxima tarefa a ser selecionada possui complexidade limitada por $\mathcal{O}(n^2)$, determinando a complexidade da heurística.

4.2.4 Algoritmo genético de chaves aleatórias viciadas

Entre as várias dificuldades decorrentes da abordagem de problemas $\mathcal{NP}$-Difíceis, escapar de ótimos locais não globais é um dos maiores desafios. Diante disso, diversas estratégias propondo alternativas genéricas para escapar de tais locais são apresentadas na literatura. Estas estratégias são classificadas como metaheurísticas e consistem da aplicação coordenada de métodos de busca visando a *intensificação* e *diversificação* de regiões do espaço de soluções por métodos de alto nível. Durante a etapa de intensificação são aplicados mecanismos genéricos com a proposta de encontrar melhores soluções. A etapa de diversificação também utiliza apenas de mecanismos genéricos e visa garantir uma exploração equilibrada das regiões do espaço de busca, incluindo regiões consideradas não promissoras. Como são desenvolvidos de forma genérica, as metaheurísticas podem ser adaptadas para resolver diferentes problemas combinatórios.

Metaheurísticas inspiradas na teoria da evolução de *Charles Darwin*, são conhecidas como Algoritmos Evolutivos. De acordo com a teoria de Darwin, os indivíduos mais adaptados possuem maiores chances de sobreviver e se reproduzir. Buscando reproduzir tal característica, estas metaheurísticas evoluem uma *população* de indivíduos ao longo de sucessivas gerações. Cada indivíduo da população representa uma solução, e cada geração é uma iteração do algoritmo. A função objetivo normalmente estabelece a *qualidade* de cada indivíduo (ou *fitness*) e é utilizada para determinar os melhores e mais adaptados

indivíduos que possuirão maiores chances de participar das próximas gerações.

Dentre os algoritmos evolutivos tem-se os Algoritmos Genéticos (AG). A principal característica desta classe de algoritmos é a utilização de *cromossomos* para representar os indivíduos de uma população. Cada cromossomo é composto por uma cadeia de valores chamados *genes*. Durante a etapa de intensificação, as população das novas gerações são criadas através da recombinação (*crossover*) dos cromossomos das gerações anteriores. Frequentemente, dois cromossomos da geração atual, chamados de pais, são *selecionados* e têm seus genes combinados, resultando em um ou mais cromossomos descendentes que serão parte da população da geração seguinte. Para diversificar a população e, consequentemente, diversificar a busca diminuindo a possibilidade de que as soluções convirjam para um ótimo local prematuramente, os cromossomos descendentes passam por um processo de *mutação*, em que o cromossomo pode ter seus genes alterados de forma aleatória.

Uma nova classe de algoritmos genéticos, denominada Algoritmos Genéticos de Chaves Aleatórias (*Random-Key Genetic Algorithms*, RKGA) foi proposta por Bean (1994). Nesta nova classe, a *codificação* utilizada é a de chaves geradas aleatoriamente no intervalo contínuo $[0, 1]$. Um exemplo de cromossomo com oito chaves aleatórias é mostrado na Figura 11.

0,251	0,787	0,374	0,755	0,589	0,451	0,075	0,576

Figura 11 – Exemplo de cromossomo com oito chaves aleatórias.

Uma *população* P no RKGA é representada por um conjunto de p cromossomos possuindo x chaves aleatórias cada. A Figura 12 mostra um exemplo de população de cromossomos em que o *fitness* é destacado (em cinza) próximo a cada cromossomo. A evolução da população ocorre aplicando-se o princípio Darwinista no qual os melhores indivíduos de uma população têm maiores chances de se reproduzir, perpetuando seu material genético nas próximas gerações. Para este fim são utilizados operadores de seleção, reprodução e mutação.

A população inicial do RKGA é gerada de maneira totalmente aleatória. A população de cada uma das próximas iterações são formadas a partir da aplicação dos operadores evolutivos na população anterior. Depois de formada, a população é separada em dois conjuntos denominados *elite* e *não-elite*. O conjunto elite, ou p_e, corresponde às melhores soluções encontradas de acordo com o *fitness*, calculado pela função de avaliação e nunca é maior que a metade da população (GONÇALVES; RESENDE; TOSO, 2012). O conjunto não-elite corresponde ao restante da população. Todos os cromossomos do conjunto elite são copiados sem alteração para a população da próxima geração, em um princípio denominado elitismo, que visa preservar os melhores indivíduos sem alterações. A Figura 13 ilustra um exemplo desta classificação dos cromossomos considerando um

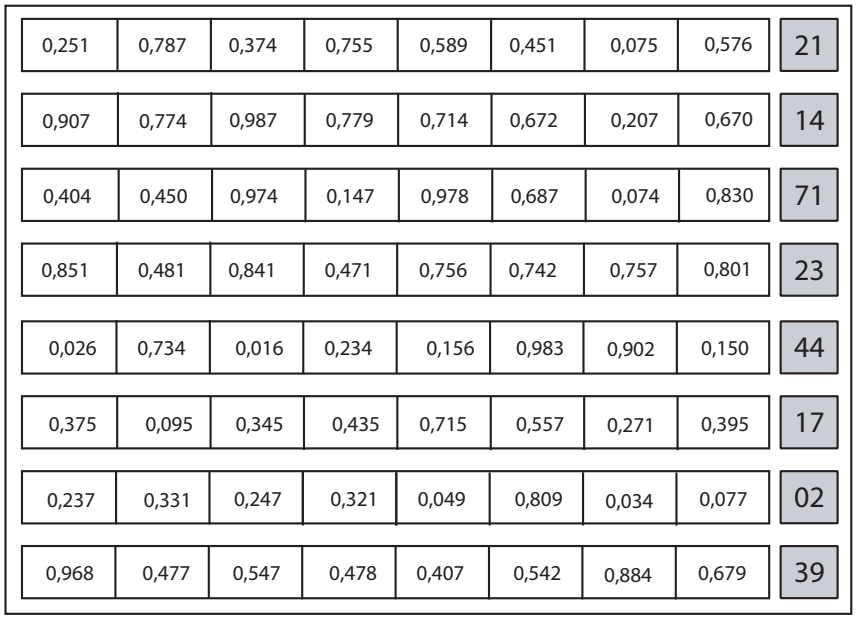


Figura 12 – Exemplo de população de cromossomos.

hipotético problema de minimização.

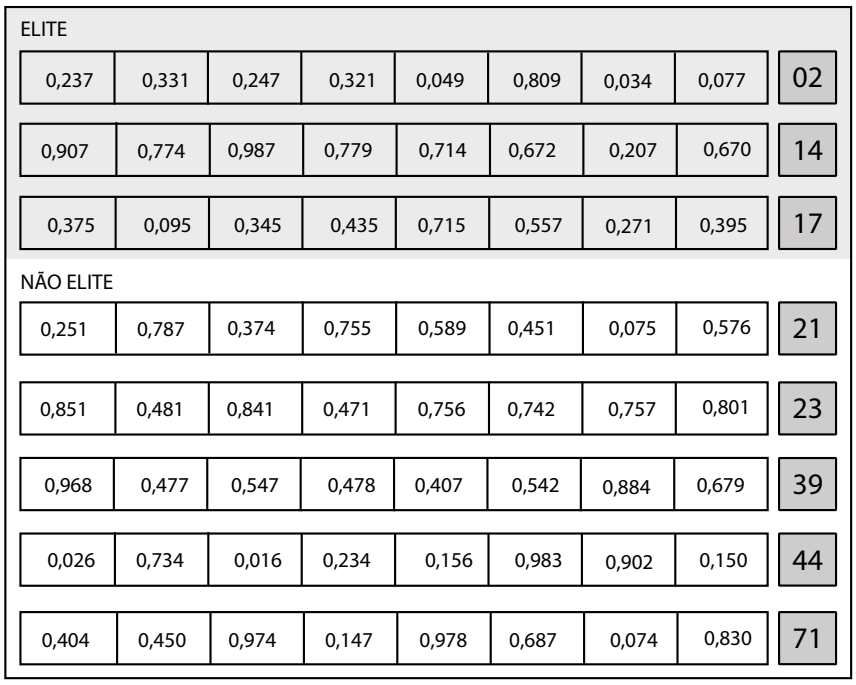


Figura 13 – Exemplo de classificação da população em elite e não-elite.

Nos algoritmos genéticos clássicos, o operador de mutação tem como função evitar a convergência prematura para um ótimo local não global por meio da alteração aleatória de pequena parte dos genes dos cromossomos já existentes. Dada a intensificação da busca ocasionada pelo elitismo, a mutação no RKGA consiste em gerar de maneira aleatória p_m cromossomos e inseri-los na população da próxima geração. Estes cromossomos são denominados *mutantes*.

O restante da população de uma geração é formado pela reprodução utilizando-se a recombinação uniforme parametrizada de Spears e Jong (1995), na qual o i-ésimo gene do cromossomo *filho* pode receber a i-ésima chave de um dos cromossomos *pais* com a mesma probabilidade. Os pares de cromossomos pais são selecionados aleatoriamente entre todos os indivíduos da k-ésima geração. A Figura 14 apresenta a construção de um cromossomo filho a partir do cruzamento dos genes de dois cromossomos pais.

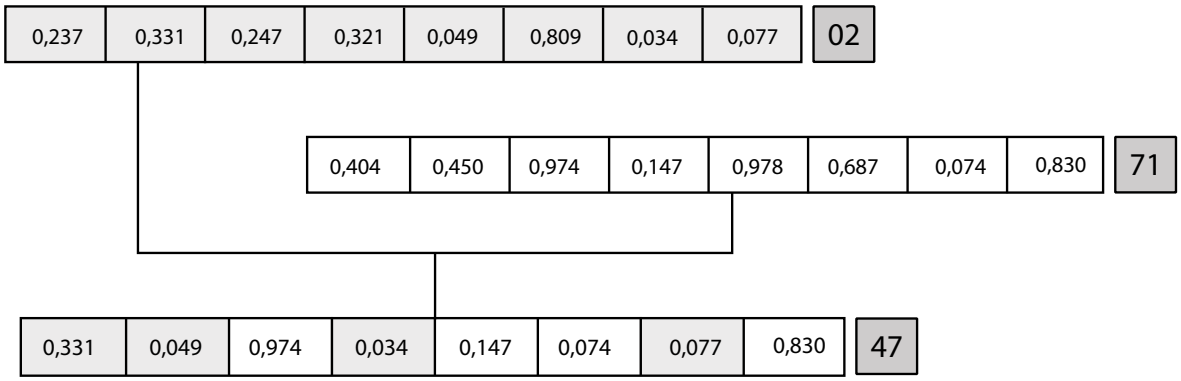


Figura 14 – Exemplo de aplicação do operador de reprodução.

O processo evolutivo repete-se até que algum critério de parada seja satisfeito. Entre os critérios de parada normalmente utilizados estão o número máximo de gerações, tempo decorrido e critérios relativos à qualidade das soluções encontradas. O processo de obtenção da solução real a partir do cromossomo, denominado *decodificação*, é dependente do problema tratado e requer conhecimento específico.

Uma variação do RKGA é apresentado por Resende (2011). Denominado Algoritmo Genético de Chaves Aleatórias Viciadas (*Biased Random-key Genetic Algorithm*, BRKGA), este algoritmo difere do RKGA na forma como os pais são selecionados e como a reprodução é implementada. No RKGA, os cromossomos pais são selecionados entre todos os indivíduos da população e o cromossomo filho possui a mesma probabilidade de herdar de ambos os pais. No BRKGA, um dos pais é sempre selecionado do conjunto elite enquanto o outro é selecionado do conjunto não-elite, além disso, o cromossomo filho tem maior probabilidade de herdar os genes do cromossomo pai pertencente ao conjunto elite. Embora a diferença entre os algoritmos seja pequena, segundo Gonçalves, Resende e Toso (2012), o BRKGA consistentemente obtém melhores soluções que o RKGA. O funcionamento geral do BRKGA é ilustrado pela Figura 15.

Para que possa ser aplicado a problemas específicos, as etapas de codificação e decodificação do BRKGA precisam ser adequadas. Adicionalmente, pode-se hibridizar o BRKGA com a utilização de buscas locais visando considerar características intrínsecas ao problema abordado no processo evolutivo. A seguir, são descritas as alterações realizadas

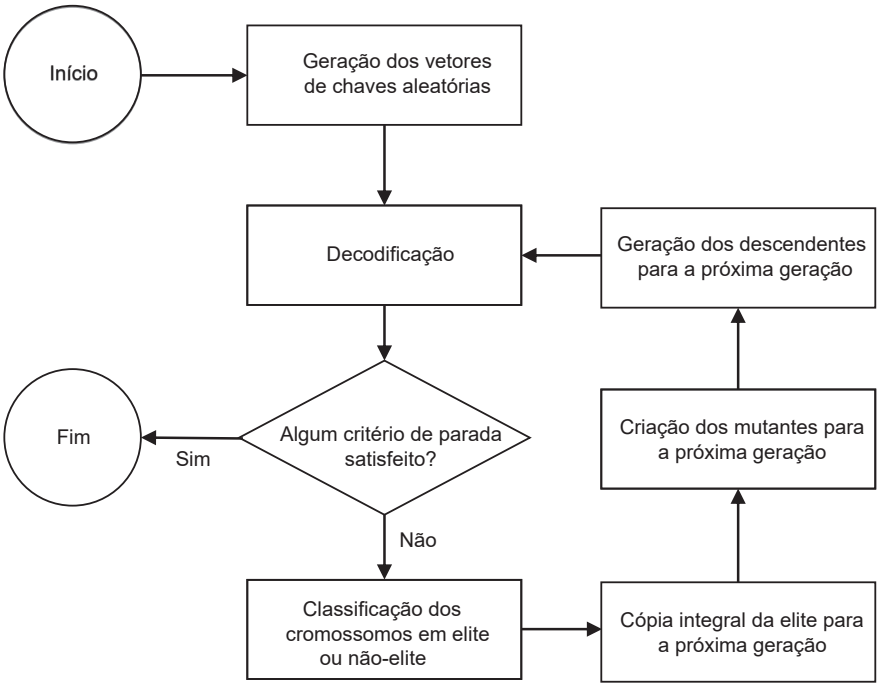


Figura 15 – Visão geral do funcionamento do BRKGA. Adaptado de Toso e Resende (2015).

ao BRKGA padrão para abordagem do RCPMS e as buscas locais utilizadas no processo de hibridização.

4.2.4.1 Codificação

De forma análoga ao RKGA, cada cromossomo do BRKGA é composto por um conjunto de x chaves aleatórias no intervalo contínuo $[0, 1] \in \mathbb{R}$. Para retratar fielmente uma solução do RCPMS, cada gene do cromossomo precisa possuir informações relacionadas às tarefas, às máquinas e ao sequenciamento das tarefas em cada máquina. Para tanto, o intervalo de valor das chaves foi alterado para $[1, m + 1) \in \mathbb{R}$, em que m é o número total de máquinas da instância. O processo para o mapeamento entre esta codificação e a informação que ela representa é detalhado na decodificação. A Figura 16 apresenta a codificação de um cromossomo válido para uma instância RCPMS contendo três máquinas e oito tarefas.

1,575	3,884	1,024	2,198	3,176	3,754	2,058	1,038

Figura 16 – Codificação de um cromossomo para uma instância do RCPMS com três máquinas e oito tarefas.

4.2.4.2 Decodificação

De acordo com Gonçalves, Resende e Toso (2012), o decodificador é um algoritmo determinístico que a partir de um cromossomo do BRKGA fornecido como parâmetro de entrada, produz uma solução para o problema de otimização abordado e o custo desta. Para decodificar o cromossomo utilizado nesta implementação do RCPMS, os respectivos genes devem ser ordenados de forma não decrescente, mantendo junto a cada gene seu índice original. Por exemplo, o cromossomo mostrado na Figura 16 é ordenado conforme visto na Figura 17.

Índice Original	2	7	0	6	3	4	5	1
Genes	1,024	1,038	1,575	2,058	2,198	3,176	3,754	3,884

Figura 17 – Cromossomo com genes ordenados em parte do processo de decodificação.

A parte inteira de cada gene representa a máquina i em que a tarefa t_j identificada pelo índice original j está alocada. A sequência de processamento das tarefas é dada por sua posição no cromossomo ordenado, ou seja, por sua parte decimal. Assim, o exemplo mostrado pela Figura 17 corresponde à alocação das tarefas 2, 7 e 0 à máquina 1, a alocação das tarefas 6 e 3 à máquina 2 e a alocação das tarefas 4, 5 e 1 à máquina 3, nesta ordem.

O tempo de processamento da máquina i é calculado somando-se o tempo ocioso gerado para esta máquina, o tempo de processamento de todas as tarefas a ela alocadas e a multiplicação do tempo de configuração para a troca de um molde pelo número de trocas necessárias. O *fitness* do cromossomo é o maior tempo de processamento entre todas as máquinas e representa o valor da solução, ou seja, o *makespan*. Como a função objetivo do RCPMS é minimizar o *makespan*, a solução para a instância será o cromossomo com menor valor de *fitness* ao final do processo evolutivo.

4.2.4.3 Critérios de parada

Neste trabalho, foram utilizados como critérios de parada do BRKGA limitantes inferiores para o valor da solução e o número máximo de gerações. Foram utilizados dois limitantes inferiores, o primeiro obtido através da modelagem descrita na Seção 4.2.1 e o segundo determinado pelo *molde dominante*, ou seja, o maior tempo necessário para processar todas as tarefas que utilizam um mesmo molde. Os demais critérios tiveram seus valores definidos nos experimentos descritos na Seção 4.3.2.

4.2.5 Buscas locais

Algoritmos de busca local definem, para uma dada solução s, uma vizinhança constituída por um conjunto de soluções com características similares. Cada solução vizinha s', é gerada a partir de um *movimento* (alteração em um ou mais elementos de s) e o valor resultante de s' é determinado considerando-se a função de avaliação. Uma solução s representa um ótimo local em relação à vizinhança adotada se todas as soluções vizinhas possuírem valor igual ou pior ao valor de s.

No intuito de intensificar a exploração realizada pelo BRKGA, são incorporadas ao processo evolutivo quatro buscas locais que consideram características intrínsecas ao RCPMS. A aplicação destas buscas locais está organizada como uma descida em vizinhança variável. A seguir, são descritas as buscas locais e a organização da aplicação das mesmas.

4.2.5.1 Busca local por inserção de tarefas

Com o intuito de maximizar o uso das máquinas que constituem o ambiente abordado e consequentemente diminuir o *makespan*, esta busca local procura balancear o tempo de processamento entre as máquinas. Para isso, movem-se as tarefas da máquina com o maior tempo de processamento (denominada *máquina crítica*) para a máquina com o menor tempo de processamento. Visando uma maior eficácia desta busca local, garantindo que ela afete também o tempo de configuração da máquina crítica, esta busca local considera as tarefas em lotes, de acordo com o molde requerido.

As máquinas que constituem uma solução são ordenadas de forma não crescente pelo tempo de processamento das máquinas. O tempo de processamento de cada molde é calculado a partir da soma dos tempos de processamento de todas as tarefas que o utilizem. As tarefas que utilizam o molde com o menor tempo de processamento são removidas da máquina crítica e inseridas de forma contígua na máquina com o menor tempo de processamento. Caso a máquina de destino utilize o molde selecionado na máquina crítica, as tarefas serão inseridas imediatamente antes da primeira tarefa que utilize este molde. Caso contrário, as tarefas serão inseridas no final do escalonamento. Após o movimento, a nova solução é avaliada e, caso possua melhor valor de solução, as máquinas são reordenadas, o valor da solução é atualizado e o processo se reinicia. O algoritmo encerra-se quando o movimento não resultar em melhora da solução.

As etapas desta busca local são apresentadas no Algoritmo 4. Como parâmetro de entrada é fornecida uma solução viável s que é utilizada como solução inicial. O laço principal (linhas 2 a 24) sistematiza a aplicação da busca local enquanto houver melhoria. Preliminarmente, não há melhoria e o conjunto de tarefas que serão removidas (J) está vazio (linhas 3 e 4). As máquinas com o maior (m_c) e menor (m_f) tempo de processamento são identificadas (linhas 5 e 6). O tempo de processamento de cada molde é calculado

pela soma dos tempos de processamento das tarefas que a utilizam. Seleciona-se o molde t com o menor tempo de processamento em m_c (linha 7). As tarefas que utilizam esse molde são removidas de m_c e adicionadas a J (linhas 8 a 12). Caso alguma tarefa em m_f utilize o molde t, as tarefas em J são inseridas imediatamente antes da primeira tarefa que o utilize. Caso contrário, as tarefas em J são inseridas ao final do escalonamento de m_f (linhas 14 a 18). A solução gerada pelo movimento é avaliada e caso ocorra melhoria no valor de avaliação, a melhoria é registrada e a solução s atualizada (linhas 19 a 22). Não havendo melhora do valor de avaliação, s é retornada e a busca local termina.

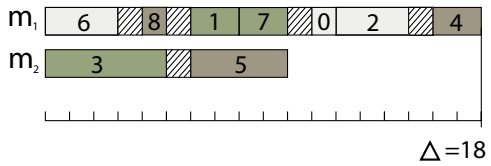
O movimento de inserção executado por esta busca local possui complexidade limitada por $\mathcal{O}(n)$. Entretanto, dada a possibilidade do movimento ocasionar intervalos ociosos em diferentes máquinas, a solução precisa ser reavaliada a cada movimento. A função de avaliação possui complexidade limitada por $\mathcal{O}(n^2)$ e determina a complexidade desta busca local.

Algoritmo 4: Busca local por inserção de tarefas.

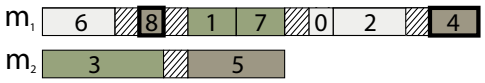
```
   Entrada: Solução s
   Saída: Solução s
1  início
2      repita
3          melhoria ← falso;
4          J ← ∅;
5          m_c ← arg max_{m∈[1,...,|M|]}(Δ_m);
6          m_f ← arg min_{m∈[1,...,|M|]}(Δ_m);
7          t ←molde com o menor tempo de processamento em m_c;
8          para cada tarefa j ∈ m_c faça
9              se j utiliza o molde t então
10                 J ← J ∪ j;
11                 retire j de m_c;
12             fim
13         fim
14         se alguma tarefa em m_f utiliza o molde t então
15             insira J em m_f antes da primeira tarefa que utiliza t;
16         senão
17             insira J em m_f após a última tarefa;
18         fim
19         se houve melhoria da solução então
20             melhoria ← verdadeiro;
21             atualize s;
22         fim
23     até melhoria = falso;
24     retorna s;
25 fim
```

A Figura 18 ilustra um movimento de inserção hipotético desta busca local. O mesmo

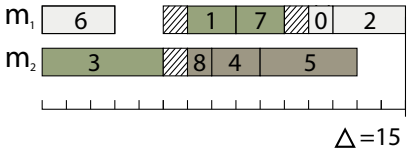
padrão estabelecido para a Figura 9 é utilizado aqui e em todas as imagens ilustrativas para as demais buscas locais. Em (a) tem-se uma dada solução viável constituída por duas máquinas. A máquina m_1 é a máquina crítica. A seleção do bloco de tarefas com o menor tempo de processamento é demonstrada em (b). Neste exemplo, o menor bloco é composto pelas tarefas 8 e 4. Em seguida, a busca local remove as tarefas pertencentes ao bloco selecionado da máquina crítica e as insere na máquina com o menor tempo de processamento (m_2). Para o exemplo apresentado, a máquina de destino possui uma tarefa (tarefa 5) que utiliza o mesmo molde requerido pelo bloco selecionado, assim, as tarefas pertencentes ao bloco são inseridas imediatamente antes da tarefa 5. A solução gerada pelo movimento e seu valor de avaliação são apresentados em (c).



(a) Solução inicial.



(b) Seleção do menor bloco de tarefas da máquina crítica.



(c) Solução gerada pelo movimento.

Figura 18 – Busca local por inserção de tarefas.

4.2.5.2 Busca local por troca de tarefas

A troca entre tarefas alocadas à máquina crítica e a uma máquina não crítica possui potencial para reduzir o *makespan* desde que o tempo de processamento das tarefas que serão inseridas na máquina crítica sejam menor do que o tempo de processamento das tarefas que serão removidas ou que a troca resulte em diminuição do tempo de configuração necessário. Com o intuito de aproveitar ao máximo tal característica, dentro do contexto abordado, esta busca local considera as tarefas em blocos, de acordo com o molde requerido, e tenta trocar o maior bloco de tarefas entre os alocados à máquina crítica com o menor bloco de tarefas alocados à máquina com o menor tempo de processamento.

Em ambas as máquinas, caso haja alguma tarefa que utilize o mesmo molde do bloco que está sendo inserido, as tarefas serão inseridas imediatamente antes da primeira tarefa que requeira tal molde. Caso contrário, as tarefas serão inseridas ao final do escalonamento. A cada movimento a nova solução é avaliada e havendo melhoria no valor de solução, as

máquinas são reordenadas, o valor da solução é atualizado e o processo se reinicia. O algoritmo encerra-se quando o movimento não resultar em melhora da solução.

As etapas desta busca local são apresentadas no Algoritmo 5. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como solução inicial. O laço principal (linhas 2 a 26) garante a aplicação da busca local enquanto houver melhoria. Inicialmente não há melhoria (linha 3). Os conjuntos de tarefas que serão removidas da máquina crítica (J_c) e da máquina com o menor tempo de processamento (J_f) estão vazios (linhas 4 e 5). As máquinas com o maior (m_c) e menor (m_f) tempo de processamento são identificadas (linhas 6 e 7). Em m_c, seleciona-se o molde (t_c) requerido pelo maior bloco de tarefas (linha 8). Todas as tarefas que utilizam este molde são removidas de m_c e adicionadas a J_c (linha 9). Em m_f, seleciona-se o molde (t_f) requerido pelo menor bloco de tarefas (linha 10). As tarefas que utilizam este molde são removidas de m_f e adicionadas em J_f (linha 11). Caso alguma tarefa em m_f utilize o molde t_c, as tarefas em J_c são inseridas imediatamente antes da primeira tarefa que o utilize. Caso contrário, as tarefas são inseridas ao final do escalonamento (linhas 12 a 16). Caso alguma tarefa em m_c utilize o molde t_f, as tarefas em J_f são inseridas imediatamente antes da primeira tarefa que o utilize. Caso contrário, as tarefas são inseridas ao final do escalonamento (linhas 17 a 21). Após o movimento, a solução gerada é avaliada. Havendo melhoria, a mesma é registrada e a solução s atualizada (linhas 22 a 25). Caso o movimento não resulte em melhoria da solução, a solução s é retornada e o algoritmo termina (linhas 26 a 28).

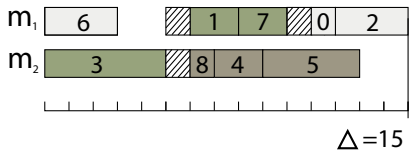
O movimento executado por esta busca local possui complexidade assintótica limitada por $\mathcal{O}(n)$. Entretanto, novamente a complexidade da busca local é determinada pela função de avaliação, ou seja, $\mathcal{O}(n^2)$.

Algoritmo 5: Busca local por troca de tarefas.

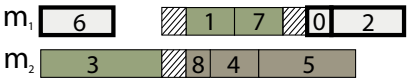
Entrada: Solução s
Saída: Solução s

início
  repita
    melhoria $\leftarrow$ **falso**;
    $J_c \leftarrow \emptyset$;
    $J_f \leftarrow \emptyset$;
    $m_c \leftarrow \arg\max_{m \in [1,\ldots,|M|]}(\Delta_m)$;
    $m_f \leftarrow \arg\min_{m \in [1,\ldots,|M|]}(\Delta_m)$;
    $t_c \leftarrow$ molde com o maior tempo de processamento em m_c;
    retire todas as tarefas que utilizam t_c de m_c e insira em J_c;
    $t_f \leftarrow$ molde com o menor tempo de processamento em m_f;
    retire todas as tarefas que utilizam t_f de m_f e insira em J_f;
    se *alguma tarefa em* m_f *utiliza o molde* t_c **então**
      insira J_c em m_f antes da primeira tarefa que utiliza t_c;
    senão
      insira J_c em m_f após a última tarefa;
    fim
    se *alguma tarefa em* m_c *utiliza o molde* t_f **então**
      insira J_f em m_c antes da primeira tarefa que utiliza t_f;
    senão
      insira J_f em m_c após a última tarefa;
    fim
    se *houve melhoria da solução* **então**
      melhoria $\leftarrow$ **verdadeiro**;
      atualize s;
    fim
  até *melhoria* = ***falso***;
  retorna s;
fim

Um movimento hipotético desta busca local é ilustrado pela Figura 19. Em (a) temos uma dada solução viável consistindo de duas máquinas. A máquina m_1 é a máquina crítica e a máquina m_2 possui o menor tempo de processamento. Em (b) é demonstrado a seleção do maior bloco de tarefas na máquina crítica, neste exemplo, o bloco é constituído pelas tarefas 6, 0 e 2. Em (c) é demonstrado a seleção do menor bloco de tarefas na máquina com o menor tempo de processamento. Neste exemplo o bloco é constituído apenas pela tarefa 3. Em seguida, os blocos são removidos de suas máquinas originais e inseridos de forma cruzada. Em m_1, o molde utilizado pelo bloco que está sendo inserido é utilizado pelas tarefas 1 e 7, assim, o bloco é inserido imediatamente antes da tarefa 1. Em m_2, o molde utilizado pelo bloco que está sendo inserido não é requerido por nenhuma tarefa, assim, o bloco é inserido ao final do escalonamento. Em (d) temos a solução gerada pelo movimento e sua avaliação.

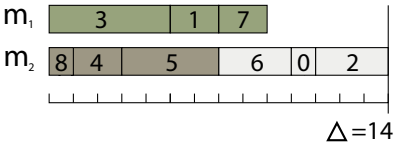


(a) Solução inicial.



(b) Seleção de bloco de tarefas na máquina crítica.

(c) Seleção de bloco de tarefas na máquina com o tempo de processamento.



(d) Solução vizinha gerada.

Figura 19 – Busca local por troca de tarefas.

4.2.5.3 Busca local por reposicionamento de tarefas

A utilização de um mesmo molde por tarefas alocadas à máquinas diferentes, em um intervalo sobreposto de tempo, resulta em um intervalo ocioso em uma das máquinas à espera pela liberação do molde requerido. Tais intervalos ociosos, quando ocorrem na máquina crítica, afetam diretamente o valor da solução. Assim, esta busca local tem como objetivo diminuir tais intervalos.

Para tanto, as máquinas são ordenadas de forma não crescente pelo tempo de processamento. Os intervalos ociosos da máquina crítica são identificados e, caso haja mais de um, são abordados na ordem em que ocorrem. Em seguida, a busca local procura entre as tarefas alocadas após o intervalo ocioso, tarefas que não utilizem o molde que gerou a ociosidade e que a soma do tempo de processamento com o tempo de configuração, considerando-se um escalonamento imediatamente posterior a tarefa que precede o intervalo ocioso, seja menor ou igual ao tamanho do intervalo.

Caso haja mais de uma tarefa candidata, seleciona-se aquela que preencha melhor o intervalo ocioso. A tarefa é posicionada imediatamente após a tarefa que precede o intervalo. A solução é reavaliada e havendo melhoria o processo se reinicia. A busca local encerra-se quando não houver movimentos que resultem em melhoria da solução. Caso o movimento executado pela busca local resulte em novos intervalos ociosos na máquina com o maior tempo de processamento, eles serão abordados nas próximas iterações da busca local. Caso sejam criados intervalos ociosos em outras máquinas, estes serão ignorados por não afetarem diretamente o valor do *makespan*.

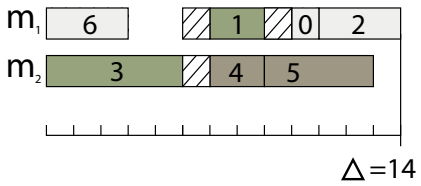
O Algoritmo 6 descreve as etapas desta busca local. Como parâmetro de entrada é fornecida uma solução viável s que é utilizada como solução atual. Entre as máquinas que

constituem a solução atual, identifica-se a aquela com o maior tempo de processamento (linha 2). Para cada intervalo ocioso existente nesta máquina, procuram-se tarefas que possam ser movidas de forma que esta lacuna seja preenchida (linhas 3 e 4). Havendo mais de uma tarefa candidata, é selecionada a tarefa que melhor preencha o intervalo (linha 5). A tarefa selecionada é movida para o início do intervalo (linha 6). Caso o valor de avaliação da solução gerada pelo movimento seja inferior ao valor de avaliação de s, a solução s é atualizada e a busca local se reinicia (linhas 7 a 9). Caso contrário, o movimento é descartado (linha 11) e o próximo intervalo ocioso, caso exista, é analisado. Não havendo melhora da solução, s é retornada (linha 15) e a busca local termina. Embora o movimento desta busca local possua complexidade linear, novamente, a necessidade de reavaliação da solução a cada movimento implica em uma complexidade quadrática.

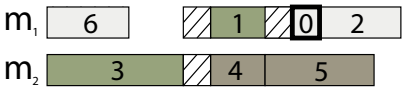
Algoritmo 6: Busca local por reposicionamento de tarefas.

```
   Entrada: Solução s
   Saída: Solução s
 1 início
 2     m_c ← arg max_{m∈[1,...,|M|]}(Δ_m);
 3     para cada intervalo ocioso o_i ∈ m_c faça
 4         se existem tarefas candidatas então
 5             t_j ← melhor candidata;
 6             mova t_j para o início de o_i;
 7             se houve melhoria da solução então
 8                 atualize s;
 9                 volta à linha 2;
10             senão
11                 desfaça o movimento;
12             fim
13         fim
14     fim
15     retorna s;
16 fim
```

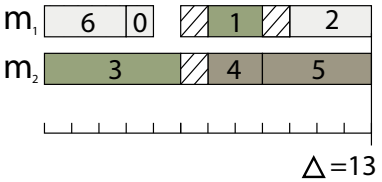
A Figura 20 ilustra um movimento hipotético desta busca local. Em (a) tem-se uma dada solução viável constituída por duas máquinas. A máquina m_1 possui o maior tempo de processamento e um único intervalo ocioso. Em (b) é demonstrada a seleção de tarefas candidatas a serem movidas para o intervalo. Neste exemplo, apenas a tarefa 0 atende aos critérios exigidos. Em seguida, conforme demonstrado em (c), a tarefa 0 é realocada para o início do intervalo e a nova solução avaliada.



(a) Solução inicial.



(b) Seleção de tarefa candidata na máquina com maior tempo de processamento.



(c) Solução vizinha gerada.

Figura 20 – Busca local por reposicionamento de tarefas.

4.2.6 Busca local por agrupamento de blocos de uns

No contexto do problema de minimização de blocos de uns, conforme visto no Capítulo 2, ao abordarmos matrizes binárias, um conjunto consecutivo de elementos com valores iguais a 1, em uma mesma linha, é conhecido como bloco de uns. Os elementos com valores iguais a 0, que separam tais blocos, são conhecidos como descontinuidades.

Fundamentados em tais conceitos, Paiva e Carvalho (2017) apresentam uma busca local visando diminuir o número de trocas de ferramentas em uma máquina flexível. O cerne desta busca local é tentar evitar que uma ferramenta a ser utilizada por uma tarefa em posição subsequente no escalonamento seja removida, o que é identificado pela ocorrência de descontinuidades em uma matriz binária. Conforme mencionado, os moldes utilizados por esta versão do RCPMS são análogos as ferramentas do IPMTC. Da mesma forma, a diminuição do número de trocas de moldes pode resultar em melhoria das soluções. Mesmo em instâncias com $\bar{p} = 0$, o agrupamento de tarefas com o mesmo molde pode resultar em melhoria da solução ao ampliar o número de moldes que podem ser utilizados em um determinado intervalo de tempo por outras máquinas.

Para a implementação desta busca local, uma matriz binária, análoga à matriz de ferramentas do SSP e IPMTC, aqui denominada *matriz de moldes*, é empregada para identificar os moldes utilizados por cada tarefa, em cada máquina. As máquinas são ordenadas de forma não crescente pelo tempo de processamento. A matriz de moldes da máquina com o maior tempo de processamento é analisada, linha a linha, em ordem aleatória a procura de descontinuidades. Caso sejam encontradas, a busca local move as colunas do bloco de uns que antecede a descontinuidade, uma a uma, para a esquerda e para a direita do bloco de uns que sucede a descontinuidade. A cada movimento a solução

é avaliada e opta-se pelo movimento que resultar no melhor valor de solução. Em caso de empate, o último movimento avaliado é mantido. Se nenhum movimento ocasionar melhoria na solução, nenhum movimento é realizado. A busca local encerra-se quando nenhum movimento ocasionar melhoria da solução.

O Algoritmo 7 ilustra as etapas desta busca local. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como solução inicial. O laço principal (linhas 2 a 23) coordena a aplicação desta busca local enquanto houver movimentos que resultem em melhoria da solução. Inicialmente, não há melhoria (linha 3). A máquina com o maior tempo de processamento é identificada (linha 4) e sua matriz sua matriz de moldes é analisada linha a linha em ordem aleatória (linhas 5 e 6). Existindo descontinuidades na matriz, a busca local move, coluna a coluna, o primeiro bloco de uns para à esquerda e à direita do segundo bloco de uns (linhas 7 a 10 e 12). Cada movimento é avaliado (linhas 11 e 13) e a coluna mantida na posição em que resultar no melhor valor de solução, considerando-se a posição original e as duas permutações (linha 14). Após a análise das permutações, havendo o algoritmo encontrado uma solução melhor que a inicial (linha 19), a solução é atualizada e o processo se reinicia (linhas 19 a 22). Não havendo permutações que melhorem a solução atual, a busca local encerra-se. O movimento para agrupar os blocos possui complexidade limitada por $\mathcal{O}(n)$. A cada movimento, a solução é reavaliada em tempo $\mathcal{O}(n^2)$, assim, a complexidade desta busca local é limitada por $\mathcal{O}(n^3)$.

Algoritmo 7: Busca local por agrupamento de blocos de uns.

Entrada: Solução s
Saída: Solução s

```
1  início
2      repita
3          melhoria ← falso;
4          m_c = arg max_{m∈[1,...,|M|]}(Δ_m);
5          Q ← matriz de moldes de m_c;
6          para cada linha r ∈ Q selecionada aleatoriamente faça
7              localize o primeiro 1-bloco i ∈ r;
8              enquanto existir um próximo 1-bloco j ∈ r faça
9                  para cada coluna k ∈ i faça
10                     e ← mova k para esquerda de j;
11                     avalie a solução gerada pelo movimento e;
12                     d ← mova k para direita de j;
13                     avalie a solução gerada pelo movimento d;
14                     posicione k onde houver o melhor valor de solução considerando
                        a posição inicial e as duas permutações;
15                 fim
16                 i ← j;
17             fim
18         fim
19         se houve melhoria da solução então
20             melhoria ← verdadeiro;
21             atualize s;
22         fim
23     até melhoria = falso;
24     retorna s;
25 fim
```

A Figura 21 ilustra um movimento hipotético desta busca local. Em (a) tem-se a matriz de moldes da máquina com o maior tempo de processamento em uma dada solução viável e o valor da solução calculado pela função objetivo do problema. Em (b) é demonstrada a seleção aleatória de uma linha da matriz para análise. A linha selecionada possui uma descontinuidade que resulta em dois blocos de uns. A busca local move as colunas do primeiro bloco de uns para a esquerda e para a direita do segundo bloco de uns. Conforme demonstrado em (c), o movimento à esquerda resulta em melhora na solução. O movimento à direita, conforme demonstrado em (d), resulta em piora da solução e por isso é descartado. Após a análise de todos os movimentos possíveis, é mantido o movimento que resultou na solução com melhor valor de avaliação, conforme visto em (c).

Moldes	Tarefas			
	6	1	0	2
1	0	1	0	0
2	1	0	1	1
3	0	0	0	0

△=14

(a) Matriz de moldes da máquina com maior tempo de processamento.

Moldes	Tarefas			
	6	1	0	2
1	0	1	0	0
2	1	0	1	1
3	0	0	0	0

△=14

(b) Seleção aleatória de linha.

Moldes	Tarefas			
	1	6	0	2
1	1	0	0	0
2	0	1	1	1
3	0	0	0	0

△=12

(c) Movimento à esquerda.

Moldes	Tarefas			
	1	0	2	6
1	1	0	0	0
2	0	1	1	1
3	0	0	0	0

△=15

(d) Movimento à direita.

Figura 21 – Busca local por agrupamento de blocos de uns.

4.2.7 Descida em vizinhança variável

Introduzido por Mladenović e Hansen (1997), a metaheurística Descida em Vizinhança Variável (ou *Variable Neighborhood Descent*, VND) baseia-se na mudança regular da estrutura de vizinhança durante a exploração do espaço de soluções. Dada uma solução inicial, as vizinhanças são exploradas de forma sequencial. A cada solução vizinha com melhor valor de avaliação, a metaheurística reinicia a exploração a partir da primeira vizinhança. Não havendo solução vizinha com melhor valor de solução, a metaheurística avança para a próxima vizinhança. A metaheurística se encerra quando um ótimo local para todas as vizinhanças for encontrado, ou seja, quando a exploração de todas as vizinhanças não resultar em solução vizinha com melhor valor de avaliação.

As etapas desta metaheurística são descritas pelo Algoritmo 8. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como solução inicial. A ordem de exploração das vizinhanças é definida por uma variável de controle (linha 2). O laço principal (linhas 3 a 22) garante a exploração das vizinhanças até que um ótimo local, comum a todas, seja alcançado. As vizinhanças são exploradas linearmente. A primeira vizinhança explorada é a busca local por inserção de tarefas (linhas 4 a 6). A seguir, são

exploradas as vizinhanças de busca local por troca de tarefas (linhas 7 a 9), busca local por reposicionamento de tarefas (linhas 10 a 12) e busca local por agrupamento de blocos de uns (linhas 13 a 15). Sempre que uma solução vizinha com melhor valor de solução é encontrada, a solução s é atualizada e a exploração se reinicia pela primeira vizinhança (linhas 16 a 18). Caso não haja melhoria da solução na vizinhança atual, move-se para a próxima vizinhança (linhas 19 e 20). Quando a exploração de todas as vizinhanças não resultar em melhoria da solução atual, a metaheurística se encerra e a solução atual é retornada (linha 23).

Algoritmo 8: Descida em vizinhança variável.

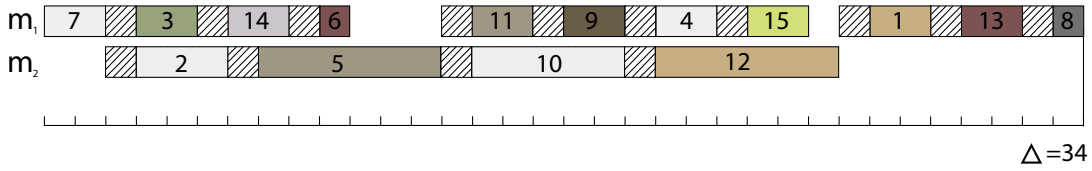
```
   Entrada: Solução s
   Saída: Solução s
 1 início
 2 |   k = 1;
 3 |   repita
 4 |   |   se k = 1 então
 5 |   |   |   s' ← execute busca local por inserção de tarefas em s;
 6 |   |   fim
 7 |   |   se k = 2 então
 8 |   |   |   s' ← execute busca local por troca de tarefas em s;
 9 |   |   fim
10 |   |   se k = 3 então
11 |   |   |   s' ← execute busca local por reposicionamento de tarefas em s;
12 |   |   fim
13 |   |   se k = 4 então
14 |   |   |   s' ← execute busca local por agrupamento de blocos de uns em s;
15 |   |   fim
16 |   |   se  valor de avaliação de s' < valor de avaliação de s então
17 |   |   |   s = s';
18 |   |   |   k = 1;
19 |   |   senão
20 |   |   |   k = k + 1;
21 |   |   fim
22 |   até k = 5;
23 |   retorna s;
24 fim
```

A Figura 22 ilustra uma aplicação hipotética do VND proposto. Em (a) tem-se uma dada solução viável com valor de avaliação igual a 34 unidades de tempo. Conforme demonstrado em (b), o VND explora a primeira vizinhança, busca local por inserção de tarefas. Neste exemplo, o menor bloco entre as tarefas alocadas à máquina crítica é composto apenas pela tarefa 8, desta forma, esta tarefa é movida de m_1 para m_2, obtendo-se uma solução com valor de avaliação igual a 32 unidades de tempo. Em nova etapa exploratória, nenhuma melhoria é registrada para esta vizinhança, assim, o VND prossegue para a próxima vizinhança, busca local por troca de tarefas. Conforme demonstrado

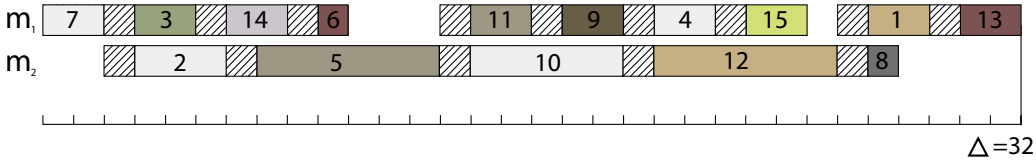
em (c), o maior bloco de tarefas na máquina crítica, composto pelas tarefas 6 e 13, é trocado pelo menor bloco da máquina com o menor tempo de processamento, composto apenas pela tarefa 8. Estes blocos são trocados resultando em uma solução com valor de avaliação igual a 31 unidades de tempo. O VND reinicia pela primeira vizinhança, que não encontra solução de melhor valor, prossegue para a segunda vizinhança que também não produz uma solução melhor, assim, explora-se a terceira vizinhança, busca local por reposicionamento de tarefas. Conforme demonstrado em (d), a máquina crítica possui dois intervalos ociosos, o primeiro é abordado e encontram-se como candidatas para realocação as tarefas 9, 4, 15 e 1. As tarefas empatam no primeiro critério de escolha, o preenchimento da lacuna ociosa, assim, é selecionada a última tarefa analisada. A tarefa 1 é reposicionada no intervalo ocioso existente entre as tarefas 14 e 11, obtendo-se uma solução com valor de avaliação igual a 30 unidades de tempo. O VND reinicia o processo de exploração linear das vizinhanças mas não encontra melhoria em nenhuma das três primeiras vizinhanças. Assim, como demonstrado em (d), o VND passa a explorar a quarta vizinhança, busca local por agrupamento de blocos de uns. A tarefa 2 é posicionada imediatamente após a tarefa 10 e uma solução com valor de avaliação igual a 26 unidades de tempo é obtida. O VND é reiniciado retomando a exploração pela primeira vizinhança. Como a exploração linear das vizinhanças não resulta em melhoria da solução atual, o VND encerra-se.

A ordem de aplicação das buscas locais foi definida de acordo com as características de cada uma, objetivando-se a maximização do efeito das mesmas sobre a solução. Inicialmente, visando corrigir soluções potencialmente desequilibradas, é aplicada a busca local por inserção de tarefas. As características desta busca local tendem a produzir um maior equilíbrio na distribuição de carga entre as máquinas que constituem a solução. Ao final da aplicação desta busca local, não há mais espaço para inserções simples. Assim, aplica-se a busca local por troca de tarefas na tentativa de efetuar inserções cruzadas entre a máquina crítica e a máquina com o menor tempo de processamento. A partir deste ponto, foca-se apenas na máquina crítica, visando diminuir seu tempo de processamento e, eventualmente, abrir espaço para novas inserções. Aplica-se então a busca local por reposicionamento de tarefas na tentativa de eliminar intervalos ociosos. Por fim, a busca local por agrupamento de blocos de uns, tenta concentrar as tarefas que utilizam o mesmo molde em um espaço contínuo de tempo, diminuindo o número de trocas de moldes e possibilitando que o mesmo molde seja utilizado por outra máquina em um intervalo diferente de tempo.

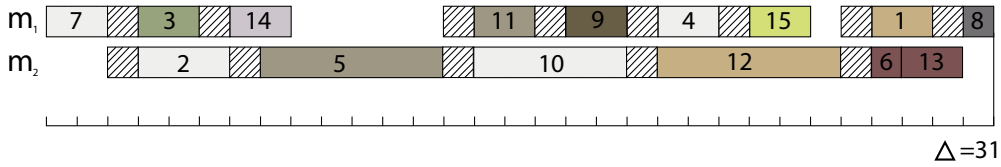
Conforme mencionado, o BRKGA copia integralmente os cromossomos pertencentes ao conjunto elite de uma geração para a outra. Sabendo-se que a aplicação das buscas locais pelo VND resulta em um ótimo local, após atingir tal ponto, os cromossomos são marcados para evitar a execução desnecessária das buscas locais em futuras gerações. Além disso, a aplicação do VND indistintamente a todos os cromossomos pode ocasionar em uma convergência prematura e aumentar significativamente o tempo de execução do



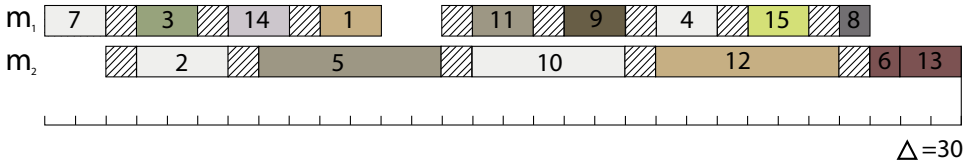
(a) Solução inicial.



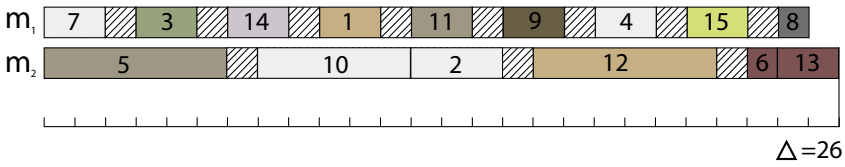
(b) Solução vizinha encontrada na primeira vizinhança.



(c) Solução vizinha encontrada na segunda vizinhança.



(d) Solução vizinha encontrada na terceira vizinhança.



(e) Solução vizinha encontrada na quarta vizinhança.

Figura 22 – Descida em vizinhança variável.

BRKGA. Assim, foram realizados experimentos preliminares que possibilitaram determinar a melhor forma de aplicação das buscas locais, fazendo-se um comparativo entre qualidade de solução e tempo de execução. Estes experimentos são descritos a seguir.

4.3 Experimentos

Uma série de experimentos computacionais foram realizados com o intuito de aferir a qualidade do método proposto e analisar seu comportamento. As seções a seguir descrevem esses experimentos e as instâncias utilizadas nos mesmos.

O ambiente computacional utilizado consiste de um computador com processador *Intel Xeon E5-2660* de 2.2 GHz com 384 GB de memória RAM sob o sistema operacional CentOS 6.8. Para a implementação do BRKGA foi utilizada a *brkgaAPI*, desenvolvida em C++ por Toso e Resende (2015), contendo as implementações de todas as fases do BRKGA padrão. O código original foi alterado para adaptação ao RCPMS, compilado com GCC 10.1.0 e a opção de otimização -O3. Os modelos descritos nas Seções 4.2.1 e 4.2.2 foram implementados utilizando-se a linguagem *Python* versão 2.7.12 e o *solver Gurobi* versão 8.1.0.

4.3.1 Conjuntos de instâncias

Considerando-se apenas o contexto de limitações originadas pelo compartilhamento de moldes, o RCPMS possui somente dois conjuntos de instâncias disponíveis na literatura, ambos propostos por Chung et al. (2019). O primeiro conjunto, contendo os menores problemas, é composto por 260 instâncias e o número de tarefas em cada instância varia entre 6, 9, 10 e 15. O segundo conjunto, contendo os maiores problemas, é composto por 320 instâncias e o número de tarefas em cada instância varia entre 20, 50, 100 e 150. Em ambos os conjuntos o número de moldes varia entre 4, 6, 8 e 10, o número de máquinas é sempre igual a 2 e o tempo de configuração para a troca dos moldes é igual a zero.

Entretanto, os baixos valores de *gap* reportados por uma heurística baseada em processamento de listas, apresentada pelos mesmo autores, em comparação à solução relaxada do problema, indicam que tais instâncias são triviais. De fato, conforme reportado na Seção 4.3.3, o BRKGA reportou soluções com valor de avaliação igual a um limite inferior trivial para todas as instâncias disponíveis na literatura. Assim sendo, para uma precisa avaliação do método proposto e progressão no estudo do problema, um novo conjunto de instâncias é apresentado.

Abordando o IPMTC, Beezão et al. (2017) apresentam dois novos conjuntos de instâncias para o problema. Nestes conjuntos, as duas principais limitações encontradas nas instâncias disponíveis para o RCPMS não ocorrem, a saber, a limitação do número de máquinas a apenas duas e a não consideração do tempo de configuração necessário para a troca dos moldes. Sendo o RCPMS um problema relacionado ao IPMTC e contendo tais conjuntos de instâncias as características desejadas, optou-se por adaptar tais conjuntos de instâncias ao RCMPS.

As características originais referentes ao número de máquinas, tempo de processamento das tarefas e tempo de configuração necessário para a troca de ferramentas foram mantidas. Para determinar o número de moldes disponíveis para cada instância, definiu-se três constantes baseadas no número de máquinas da instância, a saber, $t_0 = m + 1$, t_1 e t_2 iguais ao primeiro e segundo números primos maiores que $m + 1$, respectivamente. Para cada combinação de número de máquinas, número de tarefas e número de moldes foram

geradas cinco instâncias.

As Tabelas 12 e 13 apresentam as características dos dois conjuntos de instâncias gerados. São apresentados o número de máquinas ($|M|$), o número de tarefas ($|N|$), o número de moldes ($|T|$), o número de combinações obtidas a partir dos parâmetros anteriores, o número de instâncias selecionadas por combinação e o total de instâncias propostas.

Tabela 12 – Características do conjunto de instância RCPMS-I.

$\|M\|$	$\|N\|$	$\|T\|$	Combinações	Instâncias geradas	Total
$\{2\}$	$\{8\}$	$\{t_0, t_1, t_2\}$	3	5	15
$\{2, 3\}$	$\{15\}$	$\{t_0, t_1, t_2\}$	6	5	30
$\{2, 3, 4\}$	$\{25\}$	$\{t_0, t_1, t_2\}$	3	5	45
Total					90

Tabela 13 – Características do conjunto de instância RCPMS-II.

$\|M\|$	$\|N\|$	$\|T\|$	Combinações	Instâncias geradas	Total
$\{3, 4, 5\}$	$\{50\}$	$\{t_0, t_1, t_3\}$	9	5	45
$\{4, 5, 6, 7\}$	$\{100\}$	$\{t_0, t_1, t_3\}$	12	5	60
$\{6, 7, 8, 9, 10\}$	$\{200\}$	$\{t_0, t_1, t_3\}$	15	5	75
Total					180

Durante o processo de geração das instâncias, para evitar que problemas triviais viessem a compor o conjunto, as instâncias geradas foram avaliadas em relação a *dominância de moldes*, ou seja, qualquer instância em que a soma do tempo de processamento de todas as tarefas que utilizam um mesmo molde fosse maior ou igual a soma do tempo de processamento de todas as demais tarefas, foi descartada. Tal estratégia foi adotada visando garantir a qualidade das instâncias propostas.

4.3.2 Experimentos preliminares

Para a definição dos parâmetros do BRKGA utilizou-se o método *offline* de configuração automática de algoritmos de otimização *irace* (LÓPEZ-IBÁÑEZ et al., 2016). Dado um conjunto de instâncias do problema e um conjunto de possíveis valores para os parâmetros, o *irace* determina as melhores combinações de valores para os parâmetros. O conjunto de instâncias utilizado neste experimento é composto por 10% das instâncias novas, selecionadas aleatoriamente. A Tabela 14 demonstra os valores considerados pelo *irace* na definição dos parâmetros e os valores selecionados.

Buscando um melhor aproveitamento da arquitetura computacional atualmente disponível, os processos de decodificação e aplicação das buscas locais foram paralelizados. O número máximo de *cores* utilizados pelo BRKGA foi limitado a quatro. Ademais, foram definidas as formas de aplicação das buscas locais e os valores para os critérios de parada.

Tabela 14 – Valores considerados e selecionados pelo *irace* para definição dos parâmetros.

Parâmetro	Opções	Selecionado
Tamanho da população	{2×n, 5×n, 10×n}	10×n
Percentual população elite	{10%, 15%, 20%, 25%, 30%}	25%
Percentual população mutante	{10%, 15%, 20%, 25%, 30%}	10%
Probabilidade de herdar do pai elite	{60%, 65%, 70%, 75%, 80%, 85%, 90%, 95%}	95%

Buscando evitar um alto consumo de tempo e a convergência prematura para ótimos locais não globais, definiu-se uma probabilidade de 5% para a aplicação das buscas locais, independente do conjunto de cromossomos abordado. Estratégias diferentes para a aplicação das buscas locais nos conjuntos de cromossomos elite e não-elite foram adotadas. Para os cromossomos pertencentes ao conjunto elite, optou-se por utilizar o método de Descida em Vizinhança Variável, descrito anteriormente na Seção 4.2.7. Para os cromossomos pertencentes ao conjunto não-elite, optou-se por aplicar apenas uma vez cada uma das buscas locais, por geração. O número máximo de gerações foi definido como 1000 e acrescentou-se um gatilho que interrompe a execução do BRKGA após 300 gerações sem melhoria da solução.

4.3.3 Avaliação do método proposto

Após a realização dos experimentos preliminares, definiu-se a versão final do BRKGA. No experimento relatado nesta seção, utiliza-se como valor de referência os limites inferiores obtidos pelo modelo descrito na Seção 4.2.1 e os limites superiores obtidos por uma nova implementação da heurística baseada em processamento de listas, proposta por Chung et al. (2019). Os resultados originais publicados foram utilizados como balizamento para validar as novas implementações realizadas. Assim como na versão original, o método baseado em processamento de listas foi implementado utilizando-se a linguagem C++.

Considerando-se o *benchmark* de instâncias disponível na literatura, para o conjunto contendo os menores problemas, o modelo descrito na Seção 4.2.2 reportou soluções ótimas para todas as instâncias. Entretanto, excetuando-se dez instâncias, todos os demais valores reportados são iguais aos reportados pela solução da relaxação do problema, que desconsidera as restrições provenientes da utilização de moldes. As dez instâncias que escapam deste limite inferior trivial enquadram-se em outro limite inferior também trivial, a dominância de moldes. Assim, todas as instâncias deste conjunto possuem valores de solução triviais. Para o conjunto contendo os maiores problemas, apenas três instâncias não possuem valor de solução iguais ao valor de solução da relaxação do problema. Novamente, tais instâncias possuem dominância de molde.

Considerando-se todas as instâncias disponíveis e 10 execuções independentes, O BRKGA reportou soluções ótimas para todas as instâncias. O tempo médio de execução

do BRKGA foi de 19,86 segundos, evidenciando a facilidade do método na abordagem de tais instâncias. O desvio padrão médio reportado foi de 0,10, demonstrando a consistência do BRKGA em gerar soluções com baixa variação em execuções independentes.

De acordo com o reportado, as instâncias disponíveis na literatura mostraram-se insuficientes para avaliar de forma precisa o método proposto. Assim, conforme relatado na Seção 4.3.1, um novo conjunto de instâncias é apresentado para melhor aferir a qualidade do método proposto. Dado o ineditismo deste conjunto de instâncias, a solução da relaxação do problema, obtida pelo modelo descrito na Seção 4.2.1, é apresentada como um limite inferior e os valores obtidos pela heurística baseada em processamento de listas, descrita na Seção 4.2.3, como um limite superior.

A Tabela 15 apresenta os resultados para o conjunto RCPMS-I. As instâncias são agrupadas pelo número de máquinas (m), tarefas (n), e moldes (r). Considerando-se dez execuções independentes do BRKGA para cada instância, a tabela apresenta a média das melhores soluções encontrada (S^*), a média dos valores de solução (S), o desvio padrão (σ), e o tempo médio de execução (T), em segundos. Além disso, são apresentados a distância percentual (gap) entre a melhor solução reportada pelo BRKGA e o valor de solução da relaxação do problema ($gap_{LB}(\%)$), calculado como $gap_{LB} = \frac{S^* - LB}{LB} \times 100$ e, entre a melhor solução reportada pelo BRKGA e o limite superior ($gap_{UB}(\%)$), calculado como $gap_{UB} = \frac{S^* - UB}{UB} \times 100$. São apresentados ainda os valores médios dos limites inferiores (LB) e superiores (UB).

Referente ao limite inferior, o BRKGA reportou um gap médio de 40,60%, com um valor máximo individual de 126,17% e mínimo de 4,08%. A maior diferença foi observada entre as soluções para o subgrupo $m = 2$, $n = 8$, $r = 7$. Apesar deste grupo ser o que contém o menor número de tarefas, a diferença entre o número de tarefas e o número de moldes é a menor de todo o conjunto, o que implica em um maior número de trocas de moldes. Sabendo-se que a relaxação do problema ignora as trocas de moldes, os altos valores de gap reportados estão em acordo com o esperado. Considerando-se o limite superior, o BRKGA reportou um gap médio de -21,29%, variando entre $gaps$ individuais de -0,60% e -59,95%.

De forma geral, o BRKGA precisou de 43,41 gerações para melhorar a solução inicial em 23,22%. Entre os procedimentos de busca local, a busca local por troca de tarefas foi a que mais contribuiu para a qualidade da solução. Em média, a busca local por troca de tarefas responde por 34,63% das melhorias, a busca local por inserção de tarefas por 29,31%, a busca local por realocação de tarefas por 18,50% e a busca local por agrupamento de blocos de uns por 17,66%. O desvio padrão médio reportado foi de 3,54 (0,87%) e o tempo de execução médio de apenas 0,41 segundos, o que demonstra a consistência do método em gerar soluções com pequena variação em baixo tempo computacional.

A Tabela 16, que segue o mesmo padrão da tabela anterior, apresenta os resultados

Tabela 15 – Resultados para o conjunto de instâncias RCPMS-I.

m	n	r	BRKGA						*Bounds*	
			S^*	S	σ	T	$gap_{LB}(\%)$	$gap_{UB}(\%)$	LB	UB
2	8	3	161,80	161,80	0,00	0,10	20,83	-4,84	134,00	169,60
2	8	5	265,00	265,00	0,00	0,11	83,66	-4,99	144,40	279,00
2	8	7	287,00	287,00	0,00	0,11	119,02	-7,67	130,40	309,20
2	15	3	276,80	278,56	1,80	0,32	9,61	-4,21	253,00	288,60
2	15	5	359,80	359,80	0,00	0,29	37,13	-4,90	263,00	379,60
2	15	7	330,60	330,96	0,32	0,34	49,35	-7,21	221,20	356,80
2	25	3	391,20	391,38	0,29	0,67	11,01	-3,93	352,60	407,40
2	25	5	444,00	444,00	0,00	0,64	20,19	-5,76	370,00	471,40
2	25	7	532,60	538,50	6,58	0,75	30,62	-6,46	407,60	569,60
3	15	4	201,40	203,96	1,92	0,17	17,18	-23,55	171,60	285,20
3	15	5	217,40	219,00	0,92	0,33	38,80	-15,62	157,00	268,60
3	15	7	250,00	251,08	0,38	0,33	61,16	-19,80	155,20	314,00
3	25	4	335,20	336,56	4,30	0,04	24,70	-47,32	268,80	675,80
3	25	5	333,20	333,92	1,20	0,18	25,82	-54,85	264,60	742,40
3	25	7	330,00	335,00	6,18	0,88	37,19	-34,32	240,40	517,20
4	25	5	259,80	265,12	14,09	0,30	38,73	-50,78	187,80	546,60
4	25	7	242,40	253,64	17,75	0,70	38,26	-49,33	175,40	488,40
4	25	11	286,00	293,20	7,92	1,05	67,49	-37,73	171,40	462,00

médios para dez execuções independentes do BRKGA para o conjunto RCPMS-II. Em ralação ao limite inferior, o BRKGA reportou um *gap* médio de 34,06%, variando entre um *gap* máximo individual de 141,86% e mínimo de 1,55%. Os maiores valores médios de *gap* foram apresentados para o subgrupo $m = 10$, $n = 200$, $r = 13$, com média de 99,81%. Os menores valores foram reportados para o subgrupo $m = 4$, $n = 100$, $r = 5$ com *gap* médio de 8,82%. Em relação ao limite superior, o BRKGA reportou um *gap* médio de -42,90%, variando entre *gaps* individuais de -9,29% e -65,50%. O subgrupo $m = 6$, $n = 200$, $r = 7$ apresentou a maior diferencia, com *gap* médio de -57,59%. As menores diferenças foram reportadas para o subgrupo $m = 6$, $n = 200$, $r = 11$ com *gap* médio de -25.19%.

Em média, a convergência do BRKGA ocorreu com 377,83 gerações, a solução inicial é melhorada em 100,77% e o tempo de execução médio foi de 155,66 segundos. Entre as buscas locais, novamente, a busca local por troca de tarefas é a que mais contribuiu para a qualidade da solução. De forma geral, a busca local por troca de tarefas responde por 30,09% das melhorias, a busca local por inserção de tarefas por 27,53%, busca local por agrupamento de blocos de uns por 24,59% e a busca local por realocação de tarefas por 18,13%. O desvio padrão médio reportado para todas as instâncias deste conjunto foi de 56,76 (9,63%), demonstrando a consistência do método em gerar soluções com pequena variação em execuções independentes.

Tabela 16 – Resultados para o conjunto de instâncias RCPMS-II.

m	n	r	BRKGA						*Bounds*	
			S^*	S	σ	T	$gap_{LB}(\%)$	$gap_{UB}(\%)$	LB	UB
3	50	4	536,00	536,10	0,32	0,22	14,86	-56,44	466,80	1244,40
3	50	5	578,00	584,14	7,06	3,10	15,40	-33,57	500,60	886,80
3	50	7	595,40	601,54	10,98	3,96	22,53	-42,00	486,00	1031,40
4	50	5	424,80	440,48	21,17	3,04	16,90	-46,33	363,40	816,40
4	50	7	438,40	460,32	18,20	4,31	20,40	-48,14	364,20	884,80
4	50	11	483,00	504,98	17,48	5,43	36,66	-38,53	353,60	796,40
4	100	5	779,20	806,56	28,75	14,25	8,82	-56,68	716,20	1834,20
4	100	7	811,20	856,68	35,91	25,07	13,67	-45,24	713,60	1526,80
4	100	11	869,60	919,58	38,17	32,41	20,37	-38,08	723,00	1409,20
5	50	6	339,40	390,02	41,66	3,09	19,76	-50,79	283,60	710,60
5	50	7	366,60	413,84	42,14	4,44	22,41	-46,41	299,60	693,00
5	50	11	393,00	422,00	26,32	5,94	32,72	-46,86	296,40	751,00
5	100	6	628,20	704,00	57,13	21,28	12,45	-51,60	558,80	1342,80
5	100	7	699,00	743,94	42,47	25,91	19,88	-48,75	538,80	1426,00
5	100	11	700,60	777,60	56,08	33,67	21,86	-47,40	575,20	1360,80
6	100	7	650,40	732,30	58,23	27,05	32,76	-41,08	489,60	1145,20
6	100	11	633,60	183,54	40,12	34,27	32,83	-38,74	476,60	1043,40
6	100	13	635,60	713,12	50,78	34,66	31,31	-44,72	484,00	1180,20
6	200	7	1165,20	1325,16	113,56	248,13	19,11	-57,59	978,00	2756,80
6	200	11	1126,40	1158,02	27,14	374,47	10,51	-25,19	1019,20	1555,40
6	200	13	1217,40	1325,60	86,49	386,67	23,57	-42,18	985,80	2158,20
7	100	8	614,60	718,24	74,92	34,08	46,52	-43,46	420,00	1123,40
7	100	11	589,40	685,18	57,73	39,41	41,01	-47,94	418,60	1153,40
7	100	13	587,00	658,22	50,93	40,68	39,50	-42,84	420,60	1044,60
7	200	8	1116,40	1281,64	93,16	281,56	30,71	-49,21	855,00	2219,20
7	200	11	1061,60	1245,46	123,32	328,60	25,10	-47,32	849,00	2138,80
7	200	13	1161,80	1276,12	78,42	375,53	38,38	-43,00	840,20	2072,20
8	200	9	1152,60	1407,80	147,10	302,54	57,37	-50,12	733,40	2312,80
8	200	11	1138,20	1284,62	86,47	327,79	54,72	-36,58	736,00	1836,40
8	200	13	1055,80	1226,26	105,05	376,92	44,02	-36,80	734,00	1705,00
9	200	10	1282,40	1481,74	129,20	331,87	89,60	-41,50	677,80	2238,40
9	200	11	907,60	947,30	30,35	358,08	35,93	-25,71	664,80	1273,60
9	200	13	1006,40	1099,56	68,63	364,79	53,64	-34,72	657,20	1572,00
10	200	11	1079,00	1174,14	53,46	320,44	79,53	-31,44	600,00	1627,40
10	200	13	1202,00	1348,04	81,11	348,21	99,81	-39,22	601,80	1987,40
10	200	17	826,60	892,58	43,42	373,99	41,40	-28,06	584,20	1181,20

4.4 Conclusões

Neste capítulo foi apresentada uma nova abordagem para o RCPMS considerando restrições originadas pelo compartilhamento de moldes. O problema abordado pertence a classe de problemas $\mathcal{NP}$-difícil e possui ampla aplicação prática nas indústrias de componentes microeletrônicos. A nova abordagem consiste da implementação da metaheurística paralela BRKGA hibridizada com quatro buscas locais organizadas em um VND.

O único *benchmark* de instâncias disponíveis na literatura foi analisado e mostrou-se trivial, não oferecendo um desafio apto a aferir a qualidade do método proposto que

apresentou resultados ótimos para todas as instâncias. Assim, um novo conjunto de instâncias, contendo 270 problemas foi apresentado. Neste novo conjunto as principais deficiências encontradas no *benchmark* da literatura foram corrigidas. Os resultados reportados pelo método proposto para o novo conjunto de instâncias foram avaliados utilizando-se valores de referência obtidos por limitantes superiores e inferiores, reportando um gap médio de 36,24% e -35,69%, respectivamente.

O novo conjunto de instâncias proposto apresenta-se como um *benchmark* favorável ao avanço do estado da arte para o problema tratado, permitindo que trabalhos futuros sejam comparados utilizando-se uma mesma base de instâncias. Além disso, dada a conhecida adequação da metaheurística BRKGA a problemas permutacionais e a hibridização feita, considerando-se características específicas do problema abordado, os resultados apresentados oferecem um novo limite superior apertado o suficiente para garantir uma qualidade mínima esperada aos trabalhos futuros.

5 Conclusões

Neste trabalho de pesquisa, abordou-se de forma geral a otimização de processos produtivos em sistemas de manufatura flexível. Dois entre os principais problemas combinatórios oriundos destes sistemas, a saber, o problema de minimização de trocas de ferramentas e o problema do sequenciamento de tarefas em máquinas idênticas paralelas com restrições de ferramentas, foram detalhados visando fornecer uma base sólida para o entendimento e abordagem dos demais problemas.

No Capítulo 3 abordou-se o problema de minimização de blocos de uns, um problema $\mathcal{NP}$-difícil diretamente relacionado a dois problemas fundamentais provenientes dos sistemas de manufatura flexíveis. Para este problema foi apresentado nova representação em grafos, uma heurística construtiva que fornece uma abordagem não tradicional ao problema permutacional e, uma implementação da metaheurística *iterated local search* utilizando componentes específicos para a natureza do problema. Os experimentos computacionais realizados utilizaram o único *benchmark* de instâncias disponível na literatura e os resultados reportados foram comparados com o método representante do atual estado da arte para o problema. Foi demonstrado que a nova representação em grafos e a heurística construtiva são eficazes na geração de soluções iniciais de boa qualidade em baixo tempo computacional. Os resultados reportados pela metaheurística proposta estabeleceram novos melhores resultados para todos os nove grupos de instâncias artificiais e para quatro das cinco instâncias reais disponíveis. Foram realizadas análises estatísticas que confirmaram que a abordagem proposta superou o método representante do estado da arte para o problema. A abordagem ao problema de minimização de blocos de uns consecutivos possui potencial para avançar o estado da arte dos dois problemas fundamentais descritos no Capítulo 2, pois trata-se de um subproblema destes.

No Capítulo 4, o problema do sequenciamento de tarefas em máquinas paralelas com limitação de recursos é abordado. Dada a amplo escopo deste problema, optou-se por abordá-lo no contexto de restrições originadas pelo compartilhamento de moldes, uma aplicação industrial recorrente nas indústrias de componentes microeletrônicos. Uma implementação metaheurística paralela algoritmo genético de chaves aleatórias viciadas foi proposta abordagem ao problema. Visando introduzir características intrínsecas do problema no processo evolutivo do método proposto, optou-se por hibridizá-lo com buscas locais organizadas em um método de descida em vizinhança variável. O método foi testado frente o único *benchmark* de instâncias disponível para o problema e reportou soluções ótimas para todas as instâncias. Entretanto, ao se avaliar em profundida tais instâncias, provou-se que as mesmas eram triviais, não representando um desafio para o método proposto e dificultando o avanço no estudo do problema.

Diante de tal cenário, apresentou-se um novo conjunto *benchmark* de instâncias, contendo problemas maiores, considerando um maior número de máquinas e o tempo de configuração necessário para a troca dos moldes. Além disso, foi garantido que tais instâncias não possuam dominância de molde, o que mostrou-se o principal problema com as instâncias da literatura. Para a avaliação do método proposto foram utilizados como valores de referência um limitante superior obtido a partir de duas heurísticas baseadas em processamento de listas e um limitante inferior obtido a partir da resolução da relaxação do problema por um modelo matemático. Os resultados reportados oferecem um novo limite superior que servirá como desafio para trabalhos futuros, motivando a continuidade dos estudos deste relevante problema.

5.1 Trabalhos futuros para esta pesquisa

O amplo trabalho de revisão bibliográfica realizado para a elaboração desta pesquisa tornou evidente algumas lacunas que podem ser abordadas nos próximos capítulos. Entre os problemas já abordados, o problema de minimização de trocas de ferramentas possui como abordagem recorrente a modelagem do problema como uma instância do problema do caixeiro viajante, seja em sua forma padrão ou em sua forma de segunda ordem. Entretanto, estudos apontam que as métricas disponíveis na literatura para o cálculo das distâncias de tal modelagem são ineficientes, sendo a proposição de um algoritmo para geração dinâmica de distâncias a serem utilizadas por métodos exatos para a modelagem uma interessante contribuição para o tema. Além disso, a recente publicação do método exato para o problema de minimização de blocos de uns consecutivos pode ser aplicado em conjunto com métodos heurísticos no desenvolvimento de uma heurística matemática para os problemas relacionados a minimização de trocas de ferramentas.

Entre os diversos problemas não abordados por esta pesquisa, o escalonamento de tarefas em máquinas paralelas não-relacionadas, presente em diversos setores industriais, tais como têxteis, químicas, tintas, semicondutores, papel e fabricação de tubos, tem se mostrado uma área promissora. Apesar de possuir literatura específica limitada, o tema possui publicações recentes que reiteram sua relevância e atualidade, o qualificando como área de interesse para pesquisas futuras.

Referências

AGAPIOU, J. S. Sequence of operations optimization in single-stage multifunctional systems. *Journal of Manufacturing Systems*, Elsevier, v. 10, n. 3, p. 194–208, 1991. Citado na página 26.

AGNETIS, A. et al. Joint job/tool scheduling in a flexible manufacturing cell with no on-board tool magazine. *Computer Integrated Manufacturing Systems*, Elsevier, v. 10, n. 1, p. 61–68, 1997. Citado na página 67.

AGNETIS, A.; NICOLO, F.; LUCERTINI, M. Tool handling synchronization in flexible manufacturing cells. In: IEEE. *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. [S.l.], 1991. p. 1789–1794. Citado 2 vezes nas páginas 41 e 72.

AHMADI, E. et al. A hybrid method of 2-tsp and novel learning-based ga for job sequencing and tool switching problem. *Applied Soft Computing*, v. 65, p. 214 – 229, 2018. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494618300012>. Citado 3 vezes nas páginas 34, 36 e 37.

AIEX, R. M.; RESENDE, M. G.; RIBEIRO, C. C. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, Springer, v. 1, n. 4, p. 355–366, 2007. Citado na página 63.

AL-FAWZAN, M. A.; AL-SULTAN, K. S. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & Industrial Engineering*, v. 44, n. 1, p. 35–47, 2003. ISSN 0360-8352. Citado na página 34.

ALIZADEH, F. et al. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Algorithmica*, Springer, v. 13, n. 1, p. 52–76, 1995. Citado 2 vezes nas páginas 48 e 126.

ALLAHVERDI, A. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, Elsevier, v. 246, n. 2, p. 345–378, 2015. Citado na página 33.

ALLAHVERDI, A.; GUPTA, J.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. *Omega*, v. 27, n. 2, p. 219–239, 1999. Citado na página 33.

ALLAHVERDI, A. et al. A survey of scheduling problems with setup times or costs. *European journal of operational research*, Elsevier, v. 187, n. 3, p. 985–1032, 2008. Citado na página 33.

AMAYA, J. E.; COTTA, C.; FERNANDEZ-LEIVA, A. J. Solving the tool switching problem with memetic algorithms. *Ai Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing*, v. 26, n. 2, p. 221–235, 2012. ISSN 0890-0604. Citado na página 35.

AMAYA, J. E.; COTTA, C.; FERNáNDEZ, A. J. A memetic algorithm for the tool switching problem. In: BLESA, M. J. et al. (Ed.). *Hybrid Metaheuristics*. [S.l.]: Springer, 2008, (Lecture Notes in Computer Science, 5296). p. 190–202. ISBN 978-3-540-88438-5, 978-3-540-88439-2. Citado na página 35.

AMAYA, J. E.; COTTA, C.; FERNáNDEZ-LEIVA, A. J. Memetic cooperative models for the tool switching problem. *Memetic Computing*, v. 3, n. 3, p. 199–216, 2011. ISSN 1865-9284, 1865-9292. Citado na página 35.

AMAYA, J. E.; COTTA, C.; FERNáNDEZ-LEIVA, A. J. Cross entropy-based memetic algorithms: An application study over the tool switching problem. *International Journal of Computational Intelligence Systems*, v. 6, n. 3, p. 559–584, 2013. ISSN 1875-6891. Citado na página 35.

APPLEGATE, D. et al. *Concorde TSP solver*. 2006. Disponível em: <http://www.math.uwaterloo.ca/tsp/concorde/>. Citado na página 126.

ARANTES, M. G. et al. Sistemas flexíveis de manufatura aplicados à indústria calçados infantis de birigüi. 2004. Citado na página 22.

ATKINS, J. E.; BOMAN, E. G.; HENDRICKSON, B. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, SIAM, v. 28, n. 1, p. 297–310, 1998. Citado 2 vezes nas páginas 49 e 126.

AZEVEDO, T. N.; CARVALHO, M. A. M. Uma avaliação precisa da modelagem do problema de minimização de troca de ferramentas como o problema do caixeiro viajante. In: *Anais do XLIX Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2017. p. 1351–1362. Citado na página 34.

BARD, J. F. A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, v. 20, n. 4, p. 382–391, 1988. ISSN 0740-817X. Citado 4 vezes nas páginas 29, 30, 34 e 35.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, INFORMS, v. 6, n. 2, p. 154–160, 1994. Citado na página 80.

BECCENERI, J. C.; YANASSE, H. H.; SOMA, N. Y. A method for solving the minimization of the maximum number of open stacks problem within a cutting process. *Computers & operations research*, Elsevier, v. 31, n. 14, p. 2315–2332, 2004. Citado na página 53.

BEEZÃO, A. C. et al. Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, Elsevier, v. 257, n. 3, p. 834–844, 2017. Citado 2 vezes nas páginas 42 e 99.

BERRADA, M.; STECKE, K. E. A branch and bound approach for machine load balancing in flexible manufacturing systems. *Management Science*, INFORMS, v. 32, n. 10, p. 1316–1335, 1986. Citado na página 40.

BI, Z. M. et al. Reconfigurable manufacturing systems: the state of the art. *International journal of production research*, Taylor & Francis, v. 46, n. 4, p. 967–992, 2008. Citado na página 26.

BRETTEL, M. et al. How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective. *International journal of mechanical, industrial science and engineering*, v. 8, n. 1, p. 37–44, 2014. Citado na página 26.

BURGER, A. P. et al. Scheduling multi-colour print jobs with sequence-dependent setup times. *Journal of Scheduling*, Springer, v. 18, n. 2, p. 131–145, 2015. Citado na página 26.

CALMELS, D. The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends. *International Journal of Production Research*, Taylor & Francis, v. 57, n. 15-16, p. 5005–5025, 2019. Citado 3 vezes nas páginas 26, 27 e 33.

CARVALHO, M. A. M.; SOMA, N. Y. A breadth-first search applied to the minimization of the open stacks. *Journal of the Operational Research Society*, Taylor & Francis, v. 66, n. 6, p. 936–946, 2015. Citado 2 vezes nas páginas 50 e 51.

CATANZARO, D.; GOUVEIA, L.; LABBé, M. Improved integer linear programming formulations for the job sequencing and tool switching problem. *European Journal of Operational Research*, v. 244, n. 3, p. 766 – 777, 2015. ISSN 0377-2217. Citado 3 vezes nas páginas 35, 36 e 37.

CHAVES, A. A. et al. Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research*, v. 67, p. 174–183, 2016. ISSN 0305-0548. Citado 2 vezes nas páginas 36 e 37.

CHAVES, A. A.; YANASSE, H. H.; SENNE, E. L. F. Uma nova heurística para o problema de minimização de trocas de ferramentas. *Gestão & Produção*, v. 19, n. 1, p. 17–30, 2012. ISSN 0104-530X. Citado na página 35.

CHEN, J.-F. Unrelated parallel machine scheduling with secondary resource constraints. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 26, n. 3, p. 285–292, 2005. Citado na página 73.

CHEN, J.-F.; WU, T.-H. Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *Omega*, Elsevier, v. 34, n. 1, p. 81–89, 2006. Citado na página 73.

CHRISTOFIDES, N. *Worst-case analysis of a new heuristic for the travelling salesman problem*. [S.l.], 1976. Citado 2 vezes nas páginas 48 e 49.

CHUNG, T. et al. Minimizing the makespan on two identical parallel machines with mold constraints. *Computers & Operations Research*, Elsevier, v. 105, p. 141–155, 2019. Citado 8 vezes nas páginas 67, 68, 74, 75, 76, 78, 99 e 101.

Confederação Nacional da Indústria. *Perfil da indústria no estado de Minas Gerais*. 2017. [Online; Acesso em 25 de abril de 2020]. Disponível em: <http://perfildaindustria.portaldaindustria.com.br/estado/mg>. Citado na página 27.

Confederação Nacional da Indústria. *A indústria em números*. 2020. [Online; Acesso em 03 de abril de 2020]. Disponível em: <https://bucket-gw-cni-static-cms-si.s3.amazonaws.com/media/filer_public/aa/38/aa38bedd-e7b3-4a7b-a89d-b72986cdf0bb/industria_numeros_marco2020.pdf>. Citado na página 26.

CRAMA, Y. et al. Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, v. 6, n. 1, p. 33–54, 1994. ISSN 0920-6299, 1572-9370. Citado 10 vezes nas páginas 29, 30, 32, 34, 36, 37, 38, 39, 47 e 67.

CROES, G. A. A method for solving traveling-salesman problems. *Operations research*, INFORMS, v. 6, n. 6, p. 791–812, 1958. Citado 2 vezes nas páginas 55 e 57.

DANG, Q.-V. et al. A matheuristic for parallel machine scheduling with tool replacements. *European Journal of Operational Research*, Elsevier, v. 291, n. 2, p. 640–660, 2021. Citado na página 42.

DANIELS, R. L.; HOOPES, B. J.; MAZZOLA, J. B. An analysis of heuristics for the parallel-machine flexible-resource scheduling problem. *Annals of Operations Research*, Springer, v. 70, p. 439–472, 1997. Citado na página 72.

DANIELS, R. L.; HUA, S. Y.; WEBSTER, S. Heuristics for parallel-machine flexible-resource scheduling problems with unspecified job assignment. *Computers & operations research*, Elsevier, v. 26, n. 2, p. 143–155, 1999. Citado na página 72.

DJELLAB, H.; DJELLAB, K.; GOURGAND, M. A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, Elsevier, v. 64, n. 1, p. 165–176, 2000. Citado na página 34.

DOM, M. *Recognition, Generation, and Application of Binary Matrices with the Consecutive Ones Property*. [S.l.]: Cuvillier Gottingen, 2009. Citado na página 47.

DYSON, R.; GREGORY, A. The cutting stock problem in the flat glass industry. *Journal of the Operational Research Society*, Springer, v. 25, n. 1, p. 41–53, 1974. Citado na página 48.

EDIS, E. B.; ARAZ, C.; OZKARAHAN, I. Lagrangian-based solution approaches for a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. In: SPRINGER. *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. [S.l.], 2008. p. 337–346. Citado na página 72.

EDIS, E. B.; OZKARAHAN, I. Solution approaches for a real-life resource-constrained parallel machine scheduling problem. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 58, n. 9-12, p. 1141–1153, 2012. Citado na página 72.

FATHI, Y.; BARNETTE, K. Heuristic procedures for the parallel machine problem with tool switches. *International Journal of Production Research*, Taylor & Francis, v. 40, n. 1, p. 151–164, 2002. Citado 3 vezes nas páginas 38, 41 e 42.

FULKERSON, D.; GROSS, O. Incidence matrices and interval graphs. *Pacific journal of mathematics*, Mathematical Sciences Publishers, v. 15, n. 3, p. 835–855, 1965. Citado 2 vezes nas páginas 46 e 50.

GAO, L. et al. A production scheduling system for parallel machines in an electrical appliance plant. *Computers & industrial engineering*, Elsevier, v. 35, n. 1-2, p. 105–108, 1998. Citado na página 72.

GAREY, M. R.; GRAHAM, R. L. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, SIAM, v. 4, n. 2, p. 187–200, 1975. Citado 2 vezes nas páginas 67 e 71.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability: a guide to the theory of NP-completeness*. [S.l.: s.n.], 1979. Citado na página 26.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. [S.l.]: WH Freeman and Company, New York, 1979. Citado 2 vezes nas páginas 46 e 47.

GONÇALVES, J. F.; RESENDE, M. G.; TOSO, R. F. Biased and unbiased random-key genetic algorithms: An experimental analysis. *AT&T Labs Research, Florham Park*, 2012. Citado 3 vezes nas páginas 80, 82 e 84.

GONZÁLEZ, M. A. et al. Scatter search with path relinking for the job shop with time lags and setup times. *Computers & Operations Research*, Elsevier, v. 60, p. 37–54, 2015. Citado na página 22.

GRAHAM, R. L. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, SIAM, v. 17, n. 2, p. 416–429, 1969. Citado na página 78.

GRAHAM, R. L. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: *Annals of discrete mathematics*. [S.l.]: Elsevier, 1979. v. 5, p. 287–326. Citado 5 vezes nas páginas 23, 24, 29, 38 e 69.

GUANGDONG, H.; PING, L.; QUN, W. A hybrid metaheuristic aco-ga with an application in sports competition scheduling. In: IEEE. *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*. [S.l.], 2007. v. 3, p. 611–616. Citado na página 71.

HADDADI, S. A note on the np–hardness of the consecutive block minimization problem. *International Transactions in Operational Research*, Wiley Online Library, v. 9, n. 6, p. 775–777, 2002. Citado na página 48.

HADDADI, S. et al. Polynomial-time local-improvement algorithm for consecutive block minimization. *Information Processing Letters*, Elsevier, v. 115, n. 6-8, p. 612–617, 2015. Citado 7 vezes nas páginas 49, 58, 59, 60, 61, 62 e 63.

HADDADI, S.; LAYOUNI, Z. Consecutive block minimization is 1.5-approximable. *Information Processing Letters*, Elsevier, v. 108, n. 3, p. 132–135, 2008. Citado 4 vezes nas páginas 48, 49, 125 e 126.

HAN, M.-H.; NA, Y. K.; HOGG, G. L. Real-time tool control and job dispatching in flexible manufacturing systems. *International Journal of Production Research*, Taylor & Francis, v. 27, n. 8, p. 1257–1267, 1989. Citado na página 72.

HERTZ, A. et al. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions*, v. 30, n. 8, p. 689–694, 1998. ISSN 0740-817X. Citado na página 34.

HERTZ, A.; WIDMER, M. An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, Elsevier, v. 65, n. 1-3, p. 319–345, 1996. Citado na página 41.

HONG, T.-P.; JOU, S.-S.; SUN, P.-C. Finding the nearly optimal makespan on identical machines with mold constraints based on genetic algorithms. In: WORLD SCIENTIFIC AND ENGINEERING ACADEMY AND SOCIETY. *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*. [S.l.], 2009. Citado na página 73.

HONG, T.-P.; SUN, P.-C.; LI, S.-D. A heuristic algorithm for the scheduling problem of parallel machines with mold constraints. In: *The 7th WSEAS International Conference on Applied Computer & Applied Computational Science*. [S.l.: s.n.], 2008. Citado na página 73.

HOP, N. V.; NAGARUR, N. N. The scheduling problem of pcbs for multiple non-identical parallel machines. *European Journal of Operational Research*, Elsevier, v. 158, n. 3, p. 577–594, 2004. Citado na página 26.

HSU, W.-L.; MCCONNELL, R. M. Pc trees and circular-ones arrangements. *Theoretical computer science*, Elsevier, v. 296, n. 1, p. 99–116, 2003. Citado na página 49.

JÄGER, G.; MOLITOR, P. Algorithms and experimental study for the traveling salesman problem of second order. In: SPRINGER. *International Conference on Combinatorial Optimization and Applications*. [S.l.], 2008. p. 211–224. ISBN 978-3-540-85097-7. Citado na página 36.

JOHNSON, D. et al. Compressing large boolean matrices using reordering techniques. In: VLDB ENDOWMENT. *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. [S.l.], 2004. p. 13–23. Citado 2 vezes nas páginas 48 e 49.

KARRAY, A.; BENREJEB, M.; BORNE, P. New parallel genetic algorithms for the single-machine scheduling problems in agro-food industry. In: IEEE. *2011 International Conference on Communications, Computing and Control Applications (CCCA)*. [S.l.], 2011. p. 1–7. Citado na página 71.

KEUNG, K.; IP, W.; LEE, T. The solution of a multi-objective tool selection model using the ga approach. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 18, n. 11, p. 771–777, 2001. Citado na página 67.

KOU, L. T. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing*, SIAM, v. 6, n. 1, p. 67–75, 1977. Citado 3 vezes nas páginas 46, 47 e 48.

KOULAMAS, C. P. Total tool requirements in multi-level machining systems. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, Taylor & Francis, v. 29, n. 2, p. 417–437, 1991. Citado na página 41.

KURZ, M.; ASKIN, R. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, Taylor & Francis, v. 39, n. 16, p. 3747–3769, 2001. Citado na página 41.

LAPORTE, G.; SALAZAR-GONZALEZ, J. J.; SEMET, F. Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, v. 36, n. 1, p. 37–45, 2004. ISSN 0740-817X. Citado 3 vezes nas páginas 34, 35 e 36.

LEE, Y.-C. et al. A ga-based scheduling algorithm on parallel machines with heterogeneous mounted molds. In: IEEE. *2014 IEEE International Conference on Granular Computing (GrC)*. [S.l.], 2014. p. 147–150. Citado na página 74.

LEE, Y. H.; PINEDO, M. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, Elsevier, v. 100, n. 3, p. 464–474, 1997. Citado na página 73.

LI, D. et al. A hybrid differential evolution algorithm for parallel machine scheduling of lace dyeing considering colour families, sequence-dependent setup and machine eligibility. *International Journal of Production Research*, Taylor & Francis, p. 1–17, 2020. Citado na página 67.

LIMA, J. R.; CARVALHO, M. A. M. Descent search approaches applied to the minimization of open stacks. *Computers & Industrial Engineering*, v. 112, p. 175 – 186, 2017. ISSN 0360-8352. Citado 2 vezes nas páginas 50 e 51.

LIN, S. Computer solutions to the travelling salesman problem. bell. *System Tech. J. 44, 2245*, v. 2269, 1965. Citado na página 48.

LINHARES, A.; YANASSE, H. H. Connections between cutting-pattern sequencing, VLSI design, and flexible machines. *Computers & Operations Research*, Elsevier, v. 29, n. 12, p. 1759–1772, 2002. Citado na página 45.

LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, Elsevier, v. 3, p. 43–58, 2016. Citado 2 vezes nas páginas 60 e 100.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2019. p. 129–168. Citado na página 55.

MADSEN, O. B. Glass cutting in a small firm. *Mathematical Programming*, Springer, v. 17, n. 1, p. 85–90, 1979. Citado na página 48.

MADSEN, O. B. An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry. *Journal of the Operational Research Society*, Springer, v. 39, n. 3, p. 249–256, 1988. Citado na página 48.

MCNAUGHTON, R. Scheduling with deadlines and loss functions. *Management Science*, INFORMS, v. 6, n. 1, p. 1–12, 1959. Citado na página 37.

MECLER, J.; SUBRAMANIAN, A.; VIDAL, T. A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, Elsevier, v. 127, p. 105153, 2021. Citado na página 37.

MEERAN, S.; MORSHED, M. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of intelligent manufacturing*, Springer, v. 23, n. 4, p. 1063–1078, 2012. Citado 2 vezes nas páginas 22 e 26.

MEHRABI, M. G. et al. Trends and perspectives in flexible and reconfigurable manufacturing systems. *Journal of Intelligent manufacturing*, Springer, v. 13, n. 2, p. 135–146, 2002. Citado na página 26.

MELNYK, S. A.; GHOSH, S.; RAGATZ, G. L. Tooling constraints and shop floor scheduling: a simulation study. *Journal of Operations Management*, Wiley Online Library, v. 8, n. 2, p. 69–89, 1989. Citado na página 26.

MIKELL, P. *Automação industrial e sistemas de manufatura*. [S.l.]: Pearson Brasil, 2011. Citado 6 vezes nas páginas 11, 21, 22, 23, 24 e 25.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & operations research*, Elsevier, v. 24, n. 11, p. 1097–1100, 1997. Citado na página 95.

MOKOTOFF, E. An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research*, Elsevier, v. 152, n. 3, p. 758–769, 2004. Citado na página 75.

OHKI, M.; MORIMOTO, A.; MIYAKE, K. Nurse scheduling by using cooperative ga with efficient mutation and mountain-climbing operators. In: IEEE. *2006 3rd International IEEE Conference Intelligent Systems*. [S.l.], 2006. p. 164–169. Citado na página 71.

ÖZPEYNIRCI, S. A heuristic approach based on time-indexed modelling for scheduling and tool loading in flexible manufacturing systems. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 77, n. 5-8, p. 1269–1274, 2015. Citado na página 72.

PAIVA, G. S.; CARVALHO, M. A. M. Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers & Operations Research*, Elsevier, v. 88, p. 208–219, 2017. Citado 8 vezes nas páginas 36, 37, 42, 50, 51, 55, 57 e 92.

PINEDO, M. *Scheduling: theory, and systems*. [S.l.]: Prentice-Hall, Upper Saddle River, NJ, USA, 2008. Citado 7 vezes nas páginas 23, 24, 26, 41, 71, 75 e 76.

PINHEIRO, J. C.; ARROYO, J. E. C. Effective ig heuristics for a single-machine scheduling problem with family setups and resource constraints. *Annals of Mathematics and Artificial Intelligence*, Springer, v. 88, n. 1-3, p. 169–185, 2020. Citado na página 67.

PRIVAULT, C.; FINKE, G. k-server problems with bulk requests: an application to tool switching in manufacturing. *Annals of Operations Research*, Springer, v. 96, n. 1-4, p. 255–269, 2000. Citado na página 26.

REDDY, M. S. et al. An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. *Measurement*, Elsevier, v. 114, p. 78–90, 2018. Citado na página 67.

REINELT, G. *The traveling salesman: computational solutions for TSP applications*. [S.l.]: Springer-Verlag, 1994. Citado na página 49.

RESENDE, M. G. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. In: *Anais do XLV SBPO*. Rio de Janeiro: SOBRAPO, 2011. p. 3680–3691. Citado na página 82.

ROH, H.-K.; KIM, Y.-D. Due-date based loading and scheduling methods for a flexible manufacturing system with an automatic tool transporter. *International Journal of Production Research*, Taylor & Francis, v. 35, n. 11, p. 2989–3004, 1997. Citado na página 72.

RUIZ-TORRES, A. J.; CENTENO, G. Scheduling with flexible resources in parallel workcenters to minimize maximum completion time. *Computers & operations research*, Elsevier, v. 34, n. 1, p. 48–69, 2007. Citado na página 72.

RUIZ-TORRES, A. J.; LÓPEZ, F. J.; HO, J. C. Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. *European Journal of Operational Research*, Elsevier, v. 179, n. 2, p. 302–315, 2007. Citado na página 72.

SENNE, E. L. F. U.; YANASSE, H. H. Beam search algorithms for minimizing tool switches on a flexible manufacturing system. *Proceedings of The 11th Wseas International Conference on Mathematical and Computational Methods In Science and Engineering (MACMESE '09)*, p. 68–72, 2009. Citado na página 35.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, JSTOR, v. 52, n. 3/4, p. 591–611, 1965. Citado 2 vezes nas páginas 62 e 63.

SHIRAZI, R.; FRIZELLE, G. D. M. Minimizing the number of tool switches on a flexible machine: an empirical study. *International Journal of Production Research*, v. 39, n. 15, p. 3547–3560, 2001. ISSN 0020-7543. Citado 2 vezes nas páginas 22 e 34.

SHIVANAND, H. *Flexible manufacturing system*. [S.l.]: New Age International, 2006. Citado 3 vezes nas páginas 11, 21 e 22.

SILVA, T. T. da; CHAVES, A. A.; YANASSE, H. H. A new multicommodity flow model for the job sequencing and tool switching problem. *International Journal of Production Research*, Taylor & Francis, p. 1–16, 2020. Citado na página 36.

SOARES, L. C. et al. Heuristic methods to consecutive block minimization. *Computers & Operations Research*, Elsevier, v. 120, p. 104948, 2020. Citado na página 28.

SOARES, L. C. R.; CARVALHO, M. A. M. Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, Elsevier, v. 285, n. 3, p. 955–964, 2020. Citado na página 42.

SPEARS, W. M.; JONG, K. D. D. *On the virtues of parameterized uniform crossover*. [S.l.], 1995. Citado na página 82.

STECKE, K. E. Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, INFORMS, v. 29, n. 3, p. 273–288, 1983. Citado 4 vezes nas páginas 22, 26, 37 e 40.

STUDENT. The probable error of a mean. *Biometrika*, JSTOR, p. 1–25, 1908. Citado 2 vezes nas páginas 62 e 63.

TAMAKI, H. et al. Application of search methods to scheduling problem in plastics forming plant: A binary representation approach. In: IEEE. *Proceedings of 32nd IEEE Conference on Decision and Control*. [S.l.], 1993. p. 3845–3850. Citado 2 vezes nas páginas 72 e 73.

TANG, C. A job scheduling model for a flexible manufacturing machine. In: IEEE. *Proceedings. 1986 IEEE International Conference on Robotics and Automation*. [S.l.], 1986. v. 3, p. 152–155. Citado na página 29.

TANG, C. S.; DENARDO, E. V. Models arising from a flexible manufacturing machine, part I: Minimization of the number of tool switches. *Operations Research*, v. 36, n. 5, p. 767–777, 1988. ISSN 0030-364X. Citado 5 vezes nas páginas 29, 33, 35, 36 e 38.

THYER, G. *Computer numerical control of machine tools*. [S.l.]: Elsevier, 2014. Citado na página 21.

TOSO, R. F.; RESENDE, M. G. A c++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, Taylor & Francis, v. 30, n. 1, p. 81–93, 2015. Citado 3 vezes nas páginas 11, 83 e 99.

WIDMER, M. Job shop scheduling with tooling constraints: a tabu search approach. *Journal of the Operational Research Society*, Springer, v. 42, n. 1, p. 75–82, 1991. Citado na página 41.

XIANZHANG, C.; CHENGYAO, W. Scheduling on the parallel machines with mould constraint. In: IEEE. *SICE'99. Proceedings of the 38th SICE Annual Conference. International Session Papers (IEEE Cat. No. 99TH8456)*. [S.l.], 1999. p. 1167–1170. Citado na página 73.

XIONG, H.; FAN, H.; LI, G. Genetic algorithm-based hybrid methods for a flexible single-operation serial-batch scheduling problem with mold constraints. *Sensors & Transducers*, Citeseer, v. 155, n. 8, p. 232, 2013. Citado na página 74.

YANASSE, H. H.; LAMOSA, M. J. P. An application of the generalized travelling salesman problem: the minimization of tool switches problem. In: *International Annual Scientific Conference of the German Operations Research Society*. Bremen, Germany: [s.n.], 2006. p. 90–100. Citado na página 34.

YANASSE, H. H.; LAMOSA, M. J. P. On solving the minimization of tool switches problem using graphs. In: *XII International Conference on Industrial Engineering and Operations Management*. Fortaleza, Brazil: [s.n.], 2006. p. 1–9. Citado na página 34.

YANASSE, H. H.; RODRIGUES, R. d. C. M.; SENNE, E. L. F. Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. *Gestão & Produção*, v. 16, n. 3, p. 370–381, 2009. ISSN 0104-530X. Citado 3 vezes nas páginas 35, 36 e 37.

ZHANG, J.; HINDUJA, S. Determination of the optimum tool set for a given batch of turned components. *CIRP annals*, Elsevier, v. 44, n. 1, p. 445–450, 1995. Citado na página 26.

ZHOU, B.-H.; XI, L.-F.; CAO, Y.-S. A beam-search-based algorithm for the tool switching problem on a flexible machine. *The International Journal of Advanced Manufacturing Technology*, v. 25, n. 9-10, p. 876–882, 2005. ISSN 0268-3768, 1433-3015. Citado na página 35.

Apêndices

APÊNDICE A – Lista de publicações durante o doutorado

A.1 Artigos publicados em periódicos internacionais

Soares, L. C. R., & Carvalho, M. A. M. (2020). **Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints.** European Journal of Operational Research, 285(3), 955-964. <https://doi.org/10.1016/j.ejor.2020.02.047>

Soares, L. C. R., Reinsma, J. A., Nascimento, L. H., & Carvalho, M. A. M.(2020). **Heuristic methods to consecutive block minimization.** Computers & Operations Research, 120, 104948. <https://doi.org/10.1016/j.cor.2020.104948>

A.2 Artigos em processo de revisão

Soares, L. C. R. & Carvalho, M. A. M.(2020). **Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling in a flexible manufacturing system.** Submetido ao Computers & Operations Research em maio de 2021.

Anexos

ANEXO A – "An exact method for the consecutive block minimization" - Extraído de Soares et al. (2020)

The CBM reduction to TSP was formally proven by Haddadi and Layouni (HADDADI; LAYOUNI, 2008). The main steps of the reduction are illustrated next. Given a binary $m \times n$ matrix A, there are $\frac{n!}{2}$ unique CBM solutions owing to the reflexivity of permutations of the n columns of the matrix. The TSP, given n nodes to visit, has $\frac{(n-1)!}{2}$ unique solutions owing to the reflexivity and circular properties. Converting the CBM problem such that it has the same number of solutions as the TSP is the first part of the reduction.

Let us introduce the circular blocks minimization problem (CIR). Similar to CBM, CIR aims to minimize the number of consecutive blocks of a binary matrix. However, the first and last columns of that matrix are considered adjacent to each other, thereby granting CIR the circular property.

Next, we show how a CBM instance can be transformed to a CIR instance while preserving its optimality. Without loss of generality, matrix A will be used as an example. A column permutation $\pi_1 = [a, b, c, d, e, f]$ generates matrix A^{π_1}, which represent a solution with value of seven to CIR and eight to CBM[1].

$$A^{\pi_1} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Now, by inserting a column '0' of zeros in A, and solving the CIR again for the resulting matrix B, we may find the optimal solution $\pi_2 = [a, b, c, 0, d, e, f]$. Considering the circular property of the solution, it may be equivalently represented as $\pi_2 = [0, d, e, f, a, b, c]$. As the first column in the permutation is filled with zeros, there is no occurrence of any circular block. Consequently, π_2 is simultaneously optimal for CIR and CBM. If the zero-column is removed, the same CBM solution is found.

To reduce the CIR-transformed instance to a TSP instance, "distances" between

[1] O texto original possui um erro no exemplo. Em sua transcrição aqui, o erro foi corrigido.

pairs of columns of the matrix need to be defined to represent the distance of one node to another in a TSP tour. Different authors show that Hamming distance between columns results in perfect mapping of the solution spaces (ATKINS; BOMAN; HENDRICKSON, 1998; ALIZADEH et al., 1995), as it concerns the number of element mismatches between columns. In fact, the total tour distance of a TSP-transformed CIR instance using Hamming distance is equal to double the number of circular blocks of the CIR instance (HADDADI; LAYOUNI, 2008). An example of Hamming distance matrix D_h of binary matrix B is mentioned below.

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} 0 & a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad D_h = \begin{array}{c} \\ 0 \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{ccccccc} 0 & a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 3 & 2 & 2 & 2 & 2 & 2 \\ 3 & 0 & 3 & 3 & 3 & 5 & 3 \\ 2 & 3 & 0 & 4 & 4 & 2 & 4 \\ 2 & 3 & 4 & 0 & 4 & 4 & 2 \\ 2 & 3 & 4 & 4 & 0 & 2 & 2 \\ 2 & 5 & 2 & 4 & 2 & 0 & 4 \\ 2 & 3 & 4 & 2 & 2 & 4 & 0 \end{pmatrix} \end{array}$$

After this reduction, the Concorde Solver Applegate et al. (2006), the state-of-the-art method for solving the symmetric TSP optimally, was used to solve the resulting TSP instance. With the TSP solution generated, the CBM matrix has its columns permuted according to the tour with the zero-column being shifted to the first position of the tour and subsequently removed. Therefore, a solution for CBM is generated.

Índice