

Leonardo Cabral da Rocha Soares

**Uma Abordagem Evolucionária para o
Problema Escalonamento de Tarefas em
Máquinas Idênticas Paralelas com Limitações
de Ferramentas**

Ouro Preto

2018

Leonardo Cabral da Rocha Soares

**Uma Abordagem Evolucionária para o Problema
Escalonamento de Tarefas em Máquinas Idênticas
Paralelas com Limitações de Ferramentas**

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto para obtenção do título de Mestre 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

2018

S11a

Soares, Leonardo Cabral da Rocha .

Uma abordagem evolucionária para o problema escalonamento de tarefas em máquinas idênticas paralelas com limitações de ferramentas [manuscrito] / Leonardo Cabral da Rocha Soares. - 2018.
88f.: il.: grafs; tabs.

Orientador: Prof. Dr. Marco Antônio Moreira de Carvalho.

Dissertação (Mestrado) - Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação. Programa de Pós-Graduação em Ciência da Computação.

Área de Concentração: Ciência da Computação.

1. Máquinas-ferramenta. 2. Algoritmos genéticos. 3. Programação heurística .
I. Carvalho, Marco Antônio Moreira de . II. Universidade Federal de Ouro Preto.
III. Título.

CDU: 004.451.44



Ata da Defesa Pública de Dissertação de Mestrado

Aos 14 dias do mês de dezembro de 2018, às 14:00 horas na Sala de Seminários do DECOM no Instituto de Ciências Exatas e Biológicas (ICEB), reuniram-se os membros da banca examinadora composta pelos professores **Prof. Dr. Marco Antonio Moreira de Carvalho** (presidente e orientador), **Prof. Dr. Túlio Ângelo Machado Toffolo** e **Prof. Dr. José Elias Claudio Arroyo** aprovada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação, a fim de argüírem o mestrando **Leonardo Cabral da Rocha Soares**, com o título **"Uma Abordagem Evolucionária para o Problema Escalonamento de Tarefas em Máquinas Idênticas Paralelas com Limitações de Ferramentas"**. Aberta a sessão pelo presidente, coube ao candidato, na forma regimental, expor o tema de sua dissertação, dentro do tempo regulamentar, sendo em seguida questionado pelos membros da banca examinadora, tendo dado as explicações que foram necessárias.

Recomendações da Banca:


☒ Aprovada sem recomendações

☐ Reprovada

☐ Aprovada com recomendações: _____

Banca Examinadora:

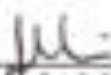

Prof. Dr. Marco Antonio Moreira de Carvalho


Prof. Dr. Túlio Ângelo Machado Toffolo

Prof. Dr. José Elias Claudio Arroyo

Certifico que a defesa realizou-se com a participação à distância do Prof. Dr. José Elias Claudio Arroyo, depois das arguições e deliberações realizadas, o participante à distância está de acordo com as recomendações da banca examinadora.


Prof. Dr. Marco Antonio Moreira de Carvalho


Prof. Dr. Joubert de Castro Lima
Coordenador do Programa de Pós-Graduação em Ciência da Computação
DECOM/ICEB/UFOP
Ouro Preto, 14 de dezembro de 2018.

Esta dissertação é dedicada à meu pai, Celso Franciso Soares, a meu professor e orientador Marco Antonio Moreira de Carvalho e a minha amada Mariana S. Torres de Oliveira. Nem uma linha teria sido escrita aqui sem o apoio destes três.

Agradecimentos

Em primeiro lugar, gostaria de agradecer a meu pai por ter aberto mão de tantas coisas ao longo da vida para me dar condições de aqui chegar. Não há palavras nem atitudes capazes de demonstrar o quão grato sou por tudo que fez e continua a fazer por mim.

Agradeço a minha amada Mariana por ter acreditado que era possível, me mostrado o caminho e por me acompanhar durante todo o processo. Seu apoio e amor foram fundamentais em todo o decorrer deste mestrado.

Agradeço a meu orientador Marco Antonio. Seu excelente trabalho como professor e orientador possibilitou todo este projeto. Sua dedicação, conhecimento e eficiência são, no mínimo, admiráveis.

Agradeço a meus todos meus colegas de mestrado, em especial a meus amigos Vinicius Gandra e Guilherme Baumgratz. Muitas foram as horas de estudos que compartilhamos e a ajuda deles foi fundamental para o sucesso desta jornada.

Gostaria ainda de agradecer a todos os demais professores e funcionários da UFOP que participaram deste processo.

À todos, meu muito obrigado.

“Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer.” (Alan Turing)

Resumo

O Problema de Escalonamento de Tarefas em Máquinas Flexíveis Paralelas Idênticas com Restrições de Ferramentas, consiste em alocar tarefas a um conjunto de máquinas flexíveis paralelas, com o objetivo de minimizar o tempo máximo de processamento das tarefas pelas máquinas. Especificamente, as tarefas possuem tempo de processamento igual em qualquer máquina, porém, possuem tempo de preparo prévio que depende de todas as tarefas anteriores sequenciadas na mesma máquina, devido a configurações de ferramentas nas máquinas flexíveis. Neste trabalho, este problema \mathcal{NP} -Difícil é abordado utilizando-se a metaheurística paralela Algoritmo Genético de Chaves Aleatórias Viciadas hibridizada com procedimentos de busca local organizados em uma Descida em Vizinhança Variável. São apresentados resultados inéditos para um conjunto de 2880 instâncias da literatura, incluindo resultados ótimos para 12,31% entre as menores instâncias. O método proposto é comparado ao atual estado da arte e obtém 91,81% das melhores soluções. Novas melhores soluções são apresentadas para 52,75% do total de instâncias. Adicionalmente, o método proposto apresenta tempo de execução 92,69% menor, dominando assim o atual estado da arte.

Palavras-chave: Escalonamento, Máquinas Flexíveis, Algoritmo Genético.

Abstract

The Identical Parallel Flexible Machines With Tooling Constraints Problem consists in scheduling tasks to a set of flexible parallel machines, with the objective of minimizing the maximum processing time of the tasks by the machines. Particularly, the tasks have the same processing time in any machine, however, they have a setup time which depends on the all of sequence of tasks scheduled on the same machine, owing to tool configurations on the flexible machine. In this work, this \mathcal{NP} -Hard problem is addressed using a parallel Biased Random-Key Genetic Algorithm hibridized with local search procedures organized in a Variable Neighborhood Descent. Results for a set of 2,880 benchmark instances from the literature are presented for the first time, including optimal solutions for 12,31% among the smallest instances. The proposed method is compared with the state of-the-art method and achieves 91,81% of the best results. New best results are presented for 52,75% of the instances. Additionally, the proposed method is 91,69% faster than the compared method, thus dominating the current state-of-art.

Keywords: Scheduling, Flexible Machines, Genetic Algorithm.

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

Figura 1 – Exemplo de solução para a instância apresentada na Tabela 2.	41
Figura 2 – Solução ótima para a instância apresentada na Tabela 2.	42
Figura 3 – Exemplo de cromossomo com seis chaves aleatórias.	46
Figura 4 – Exemplo de população de cromossomos.	46
Figura 5 – Exemplo de classificação da população em elite e não-elite.	47
Figura 6 – Exemplo de aplicação do operador de reprodução.	48
Figura 7 – Visão geral do funcionamento do BRKGA. Adaptado de Toso e Resende (2015).	48
Figura 8 – Codificação de um cromossomo para uma instância do IPMTC com duas máquinas e seis tarefas.	51
Figura 9 – Cromossomo com chaves ordenadas, em parte do processo de decodificação.	52
Figura 10 – Busca local por inserção de tarefas.	56
Figura 11 – Busca local por troca de tarefas.	59
Figura 12 – Busca local por agrupamento de blocos de uns.	62
Figura 13 – Descida em vizinhança variável.	64
Figura 14 – Gráfico <i>boxplot</i> da análise de convergência média do BRKGA.	78
Figura 15 – Gráficos ttt ilustrativos para 6 instâncias.	79

Lista de tabelas

Tabela 1	– Exemplo de ferramental por tarefa.	26
Tabela 2	– Exemplo de instância do IPMTC.	40
Tabela 3	– Exemplo de matriz de ferramentas.	40
Tabela 4	– Exemplo do plano de troca de ferramentas.	43
Tabela 5	– Resultados obtidos pelo modelo M_1 para o IPMTC-I.	68
Tabela 6	– Comparação entre as implementações do ALNS.	70
Tabela 7	– Valores considerados e selecionados pelo <i>irace</i> para definição dos parâ- metros.	71
Tabela 8	– Resultados para o grupo IPMTC-I.	73
Tabela 9	– Resultados para o grupo IPMTC-II.	75
Tabela 10	– Comparação entre o tempo de processamento do BRKGA paralelo, BRKGA sequencial e ALNS.	77

Lista de abreviaturas e siglas

AG	Algoritmo Genético
ALNS	<i>Adaptative Large Neighborhood Search</i>
BRKGA	<i>Biased Random-key Genetic Algorithm</i>
CLIP	<i>Composite Local Improvement Procedure</i>
CSTR	<i>Constructive Heuristic</i>
IPMTC	<i>Identical Parallel Machines With Tooling Constraints</i>
KTNS	<i>Keep Tool Needed Soonest</i>
LIP	<i>Local Improvement Procedure</i>
LPT	<i>Longest Processing Time</i>
MSG	<i>Multiple-Start Greedy</i>
MTSP	<i>Minimization of Tool Switches Problem</i>
RKGA	<i>Random-key Genetic Algorithms</i>
RLPP	<i>Random List Processing Procedure</i>
SMF	Sistema de Manufatura Flexível
SPM	<i>Scheduling Problem Of Parallel Machines</i>
TSP	<i>Traveling Salesman Problem</i>
VND	<i>Variable Neighborhood Descent</i>

Lista de símbolos

Δ	<i>Makespan</i>
Δ_m	Tempo de processamento da máquina m
L	Conjunto de ferramentas
M	Conjunto de máquinas flexíveis idênticas paralelas
T	Conjunto de tarefas a serem alocadas

Sumário

1	INTRODUÇÃO	25
1.1	Justificativa	29
1.2	Objetivos	29
1.3	Organização do texto	30
2	REVISÃO BIBLIOGRÁFICA	31
3	FUNDAMENTAÇÃO TEÓRICA	39
3.1	Escalonamento de tarefas em máquinas idênticas paralelas com li- mitações de ferramentas	39
3.1.1	Formulação matemática de Beezão et al. (2017)	43
3.2	Algoritmo genético de chaves aleatórias viciadas	45
4	DESENVOLVIMENTO	51
4.1	Codificação	51
4.2	Decodificação	51
4.3	Funções de avaliação	52
4.3.1	Avaliação completa	52
4.3.2	Avaliação δ	53
4.4	CrITÉRIOS de parada	53
4.5	Buscas locais	54
4.5.1	Busca local por inserção de tarefas	54
4.5.2	Busca local por troca de tarefas	56
4.5.3	Busca local por agrupamento de blocos de uns	60
4.5.4	Descida em vizinhança variável	62
5	EXPERIMENTOS COMPUTACIONAIS	67
5.1	Conjuntos de instâncias	67
5.2	Experimentos preliminares	67
5.2.1	Formulação matemática para o IPMTC	68
5.2.2	Configuração e validação da implementação do ALNS	69
5.2.3	Ajuste de parâmetros do BRKGA	71
5.3	Comparação com o estado da arte	72
5.3.1	Experimentos adicionais	76
5.4	Análise de convergência do BRKGA	77
6	CONCLUSÃO	81

REFERÊNCIAS 83

1 Introdução

Embora o significado original de *manufatura* em latim seja *feito à mão*, o termo sempre foi utilizado para descrever a transformação de matéria prima em produtos acabados, não importando se tal transformação tenha ocorrido manualmente ou com a utilização de máquinas. Com a revolução industrial, houve grandes avanços no modo de produção. A utilização de máquinas e o trabalho em série possibilitaram ganhos de produtividade, e o termo manufatura passou a ser utilizado para designar as fábricas ou indústrias em que a matéria prima era transformada em produtos.

A partir de 1914 o modelo de produção *Fordista* foi amplamente adotado por indústrias automobilísticas, têxteis, siderúrgicas e energéticas, predominando até 1970. Neste modelo de produção, as máquinas eram dispostas em uma sequência lógica e os produtos montados em esteiras rolantes que passavam pelas máquinas na sequência estabelecida. Os operários ficavam localizados ao longo da linha de montagem e executavam tarefas específicas, não sendo mais necessário mão de obra altamente capacitada. Um problema crucial do *Fordismo* era a falta de flexibilidade (WOOD JR, 1992) quanto à variedade dos produtos fabricados, uma vez que as linhas de produção eram extremamente específicas.

Após a Segunda Guerra Mundial, a fábrica automobilística Toyota estava determinada a produzir em larga escala. Em 1950, Eiji Toyoda visitou as instalações da Ford em Detroit durante três meses para observar o modelo de produção Fordista. De volta ao Japão, Eiji Toyoda e Tqiichi Ohno, especialista em produção, refletiram sobre o observado na Ford e concluíram que o sistema precisava ser adaptado para obter sucesso no Japão.

O mercado doméstico japonês era pequeno e exigia uma grande variedade de tipos de produtos, além disso, os trabalhadores japoneses não se adaptariam ao conceito de trabalho fordista. Ao reformular a linha de produção, Toyoda e Ohno desenvolveram técnicas que possibilitavam a diminuição do tempo gasto para alterar as máquinas permitindo que produtos com características diferentes pudessem ser produzidos de forma simples e rápida. Tornou-se mais barato fabricar pequenos lotes do que grandes lotes homogêneos. Surgia o conceito de produção flexível (WOOD JR, 1992).

Sistemas de Manufatura Flexíveis (SMFs) são constituídos por uma rede de máquinas flexíveis interligadas por um sistema automático de manuseio de materiais, seja matéria-prima ou produtos semiacabados. Estas máquinas são capazes de executar um grande número de operações diferentes como cortar, furar, lixar, entre outras (ZEBALLOS, 2010). 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. De maneira ampla, uma linha de produção dotada de máquinas flexíveis substitui uma ou mais linhas dedicadas, possibilitando à indústria possuir um portfólio de produtos maior a partir da adequação de suas linhas de produção.

Refere-se de maneira genérica a estas operações sobre a matéria prima ou produtos semiacabados como o *processamento* de *tarefas* por máquinas flexíveis. Cada tarefa requer um conjunto de ferramentas para o seu processamento. 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.

Para que tais trocas de ferramentas possam ocorrer, é necessário parar a linha de produção. A máquina deve ser parada, 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. Todas as ferramentas são consideradas uniformes, podendo ser alocadas em qualquer posição dentre as disponíveis no *magazine* e são trocadas em tempo constante.

A título de exemplo, uma máquina com capacidade para armazenar dez ferramentas no *magazine* deve processar as tarefas *A*, *B* e *C* nessa ordem. Cada tarefa precisa de quatro ferramentas diferentes para ser processada, como descrito na Tabela 1.

Tabela 1 – Exemplo de ferramental por tarefa.

Tarefa	Ferramentas
A	1, 2, 3, 4
B	5, 6, 7, 8
C	9, 10, 11, 12

O total de ferramentas necessárias (12) excede a capacidade do *magazine* da máquina. Tendo a configuração inicial da máquina colocado no *magazine* as ferramentas de um a dez, a máquina será capaz de processar as tarefas *A* e *B*, mas precisará trocar duas ferramentas antes de processar a tarefa *C*.

O sequenciamento do processamento das tarefas na máquina, e a determinação do plano de trocas de ferramentas são dois problemas que compõem o problema combinatório conhecido na literatura como Problema de Minimização de Trocas de Ferramentas (ou *Minimization of Tool Switches Problem*, MTSP). O primeiro problema é NP-Difícil (CRAMA et al., 1994), significando que não se conhece algoritmo determinístico polinomial para sua

solução. 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 \(1988\)](#).

Conforme apresentado por [Graham et al. \(1979\)](#), o problema de sequenciamento de tarefas em máquinas pode ser descrito 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 minimizado e geralmente contém apenas uma entrada. Segundo [Pinedo \(2008\)](#), alguns dos ambientes de máquinas possíveis para α são:

- Máquina única (1): É o caso mais simples de todos os ambientes possíveis. Apenas uma máquina processa todas as tarefas;
- Máquinas idênticas paralelas (P_m): Existem m máquinas idênticas em paralelo. Uma tarefa pode ser processada em tempo idêntico por qualquer uma das m máquinas;
- Máquinas paralelas com velocidades diferentes (Q_m): Existem m máquinas em paralelo com diferentes velocidades. A velocidade da máquina i é denotada por v_i e independe das tarefas processadas. O tempo p_{ij} que o processamento da tarefa j requer na máquina i é calculado em função de v_i ;
- Máquinas não relacionadas em paralelo (R_m): Existem m máquinas idênticas em paralelo. A máquina i pode processar a tarefa j na velocidade v_{ij} , em que a velocidade é dependente da tarefa a ser processada;
- *Flow shop* (F_m): Existem m máquinas em série. Cada tarefa precisa ser processada pelas m máquinas, seguindo uma rota pré-estabelecida. Após terminado o processamento da tarefa j na máquina i , a tarefa prossegue para a fila de processamento da máquina $i + 1$;
- *Job shop* (J_m): Em um ambiente com m máquinas, cada tarefa possui uma rota pré-determinada à seguir que passa por todas as máquinas pelo menos uma vez.

Algumas restrições e características do processamento especificadas para o campo β são:

- Tempo de disponibilização (r_j): A tarefa j só pode ser processada após decorrido o tempo r_j . Se r_j não aparece em β então a tarefa j pode ser processada a qualquer momento;

- Preempção (*prmp*): Implica a não necessidade de processamento de uma tarefa ser ininterrupto. O processamento de uma tarefa pode ser interrompido para que outra tarefa, ou parte dela, seja alocada à máquina;
- Limitações de precedência (*prec*): Há uma ordem de processamento específica a ser seguida. Uma ou mais tarefas devem estar completamente processadas antes que outra tarefa possa iniciar seu processamento;
- Tempos de preparação dependente da sequência (s_{jk}): Entre o processamento consecutivo das tarefas j e k , podem ser necessárias operações de preparação da máquina, como manutenção ou mesmo troca de ferramentas. Este campo indica que o tempo de preparação da máquina depende da sequência de processamento das tarefas. Se o tempo de configuração entre as tarefas j e k dependem da máquina onde estão alocadas, então o índice i é acrescentado, i.e., s_{ijk} .

Algumas funções objetivo comuns, especificadas pelo campo γ incluem:

- *Makespan* (C_{max}): Definido como $\max(C_1, \dots, C_n)$ é equivalente ao maior tempo de processamento total entre todas as máquinas do ambiente;
- Atraso Máximo (L_{max}): Definido como $\max(L_1, \dots, L_n)$, em um ambiente em que cada tarefa possui uma data de vencimento, o atraso máximo corresponde a maior violação da data de vencimento;
- Tempo Total de Conclusão Ponderado ($\sum w_j C_j$): A soma dos tempos de conclusão total das n tarefas considerando-se o peso um peso para cada uma.

O escalonamento de tarefas em um ambiente composto por máquinas paralelas constitui o denominado Problema de Escalonamento em Máquinas Paralelas (*Scheduling Problem of Parallel Machines*, SPM), (MCNAUGHTON, 1959), cujo objetivo é minimizar o *makespan*. Da junção do SPM com o MTSP, em que o tempo de preparação dependente da sequência está relacionado com trocas de ferramentas, surge o *Problema de Escalonamento de Tarefas em Máquinas Paralelas Idênticas com Restrições de Ferramentas* (*Identical Parallel Machines With Tooling Constraints*, IPMTC), abordado primeiramente por Stecké (1983) e objeto de estudo desta dissertação.

Para abordagem do IPMTC, propõe-se nesta dissertação a implementação da metaheurística paralela Algoritmo Genético de Chaves Aleatórias Viciadas hibridizado com três buscas locais aplicadas utilizando-se o método de Descida em Vizinhança Variável.

1.1 Justificativa

Segundo [Beezão et al. \(2017\)](#), o deslocamento gerado pela movimentação das ferramentas entre o depósito e o *magazine* da máquina representa entre 25% e 30% dos custos fixos e variáveis de um SMF e consome mais tempo que qualquer outra atividade de configuração. Portanto, diminuir tais trocas e, conseqüentemente, o tempo total de processamento das máquinas possibilita às indústrias de manufatura flexível reduzir consideravelmente seus custos operacionais. Em ambientes industriais de manufatura flexível, a utilização de máquinas paralelas idênticas é recorrente ([PINEDO, 2008](#)). Assim, o IPMTC é um problema de aplicação prática na indústria. Por tratar-se de um problema \mathcal{NP} -Difícil ([CRAMA et al., 1994](#)), seu estudo possui também grande importância teórica. Além disso, conforme relatado por [Calmels \(2018\)](#), o número de publicações em temas correlatos ao IPMTC cresce anualmente desde 1988 demonstrando a relevância e atualidade do tema.

As principais motivações para a escolha da metaheurística Algoritmo Genético de Chaves Aleatórias Viciadas foram (i) a flexibilidade que esta metaheurística possui em lidar com diversos problemas complexos de otimização combinatória ([RESENDE, 2011](#)); (ii) os bons resultados obtidos em diversos problemas de otimização ([GONÇALVES; MENDES; RESENDE, 2005](#); [GONÇALVES; RESENDE, 2011](#); [VALENTE et al., 2011](#)); e (iii) não haver trabalhos anteriores que proponham a utilização desta metaheurística na resolução do problema abordado por esta dissertação.

1.2 Objetivos

Esta dissertação propõe o estudo e abordagem computacional do Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas no contexto de sistemas de manufatura flexíveis. A abordagem se dá pela metaheurística paralela Algoritmo Genético de Chaves Aleatórias Viciadas combinada com métodos de buscas locais. São objetivos específicos:

- Realizar pesquisa para geração de embasamento teórico sobre o Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas e sobre Algoritmos Genético de Chaves Aleatórias Viciadas;
- Gerar uma revisão bibliográfica atual e abrangente para o problema tratado;
- Propor um método computacional consistente para a resolução do problema tratado;
- Analisar os resultados obtidos pelo método implementado considerando-se as instâncias disponibilizadas na literatura e compara-los com o estado da arte.

1.3 Organização do texto

O texto desta dissertação encontra-se organizado como descrito a seguir. O Capítulo 2 apresenta a revisão da literatura acerca do IPMTC. O Capítulo 3 apresenta a descrição detalhada do problema tratado e do método empregado em sua abordagem. No Capítulo 4 o método proposto como solução é detalhado, bem como a hibridização proposta. Os resultados dos experimentos computacionais são apresentados no Capítulo 5 e, por último, no Capítulo 6 são apresentadas as conclusões e os possíveis trabalhos futuros.

2 Revisão bibliográfica

Por tratar-se de um problema presente em indústrias de manufatura flexível, o IPMTC tem sido estudado desde o início da década de 1980, embora trabalhos relacionados ao escalonamento de tarefas em máquinas paralelas datem do final da década de 1950. Desde então, diversos autores têm abordado o IPMTC e seus diversos temas correlatos ([ALLAHVERDI; GUPTA; ALDOWAISAN, 1999](#); [ALLAHVERDI et al., 2008](#); [ALLAHVERDI, 2015](#); [CALMELS, 2018](#)). Em especial, [Calmels \(2018\)](#) apresenta um *survey* atualizado contendo uma classificação abrangente das principais publicações ao longo dos anos, as principais tendências de pesquisa sobre o tema e as áreas menos estudadas. Neste capítulo, apresenta-se em ordem cronológica as principais contribuições para o tema.

Segundo [Stecke \(1983\)](#), no início da década de 1980 aproximadamente 50% de todo investimento em manufatura nos Estados Unidos era destinado às indústrias que trabalhavam com metal, em especial, indústrias de corte em metal. Cerca de 75% desse investimento era gasto na manufatura de lotes com menos de 50 peças e a baixa produtividade desses sistemas preocupava a indústria. Como resposta a essa preocupação, novos métodos de produção em massa foram desenvolvidos e melhorias foram feitas na tecnologia de produção, em especial na utilização de máquinas numericamente controladas. Os estudos apontavam para um grande crescimento dos SMFs.

Neste cenário, [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 podem ser processados sequencialmente. 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. Os autores concentram-se nos problemas de agrupamento de máquinas e carregamento de tarefas e ferramentas.

Para o problema de agrupamento de máquinas o mesmo trabalho apresenta uma adaptação da proposta anterior feita por [Stecke e Solberg \(1982\)](#). A adaptação consiste

em considerar o tempo de processamento das tarefas, as limitações referentes a exigência de ferramentas pelas tarefas e a capacidade limitada do *magazine* das máquinas. Para o problema de carregamento de tarefas e ferramentas foram apresentados seis possíveis objetivos:

1. Equilibrar o tempo de processamento das máquinas;
2. Maximizar o número de tarefas consecutivas em cada máquina;
3. Equilibrar a carga de trabalho por máquina em grupos de máquinas de igual tamanho;
4. Desbalancear a carga e trabalho por máquina em grupos de máquinas de tamanhos diferentes;
5. Preencher o *magazine* o mais densamente possível;
6. Maximizar a quantidade de tarefas prioritárias.

De acordo com o problema, alguns objetivos podem ser contraditórios enquanto outros podem ser igualmente aplicados. Assim, a definição de quais objetivos deverão ser utilizados é dependente do problema a ser tratado. Como quase todos os objetivos e restrições são não lineares, foram elaboradas formulações de Programação Não Linear Inteira Mista. Cinco diferentes formas de linearização foram consideradas para solucionar as formulações propostas.

Os experimentos foram realizados considerando-se os dados da *Caterpillar Tractor Company*, uma indústria que realiza operações de corte em metal. O SMF da *Caterpillar* consistia em nove máquinas divididas em três grupos, o primeiro contendo quatro, o segundo três e o último duas máquinas. Embora os resultados alcançados tenham sido ótimos, os autores fizeram ressalvas sobre a aplicação dos métodos em instâncias maiores.

A combinação dos dois problemas abordados por [Stecke \(1983\)](#) fornecem uma definição muito próxima da atualmente utilizada para o IPMTC. Com efeito, cada um dos três grupos de máquinas utilizados no experimento computacional representa uma instância do IPMTC.

Posteriormente, [Berrada e Stecke \(1986\)](#) consideraram o mesmo contexto industrial, porém, focando exclusivamente no problema de carregamento de tarefas e ferramentas. Dentre os seis objetivos apresentados por [Stecke \(1983\)](#) para o problema, foi considerado o equilíbrio da carga de trabalho entre as máquinas de um mesmo grupo. Foi proposto um método *branch-and-bound* para solução do problema. O experimento computacional realizado utilizou dois conjuntos de instâncias. O primeiro conjunto contém um total de 10 instâncias possuindo cada uma entre 6 e 9 máquinas e variando o número de tarefas a serem alocadas entre 7 e 48. O segundo conjunto possui 45 instâncias contendo cada

uma entre 4 e 15 máquinas não idênticas e de 10 a 40 tarefas a serem alocadas. O método proposto apresentou melhores resultados para as instâncias com máquinas não idênticas devido a menor similaridade entre as soluções.

Koulamas (1991) apresentou um estudo com o objetivo de minimizar o número de trocas de ferramentas em todos os níveis de um cenário *flow shop* flexível sem aumentar o tempo total de processamento. Além das trocas de ferramentas devidas às exigências das tarefas, também considerou-se a troca de ferramentas devido ao fim da vida útil das mesmas. No contexto de indústrias de corte em metal, as trocas de ferramentas ocorriam em média 10 vezes mais pelo fim da vida útil do que por requisitos provenientes das trocas entre tarefas. Nos métodos propostos, o enfoque foi aumentar a vida útil das ferramentas, ao invés de minimizar o número de trocas pelo sequenciamento das tarefas. O experimento computacional utilizou três instâncias possuindo 4, 8 e 15 níveis de produção. Todas as instâncias possuíam 50 tarefas. O experimento atingiu as soluções ótimas com baixo tempo de processamento.

Embora o ambiente de produção deste último trabalho seja um *flow shop* flexível, Pinedo (2008) mostra que cada nível deste ambiente é uma generalização do ambiente de máquinas idênticas paralelas. Acrescentando-se a esta generalização as limitações impostas pela necessidade de troca de ferramentas e, desconsiderando-se as trocas de ferramentas devido ao fim da vida útil das mesmas, cada nível deste *flow shop* flexível pode ser considerado uma instância do IPMTC.

Outro ambiente que também é considerado uma generalização do ambiente de máquinas idênticas paralelas segundo Pinedo (2008) é o *job shop*. Hertz e Widmer (1996) apresentam um método baseado em *busca tabu* com o objetivo de minimizar o *makespan* em um ambiente *job shop* com limitações de ferramentas. O experimento computacional realizado considerou 8 problemas com o número de tarefas e máquinas variando entre 1 e 10. Os resultados foram comparados com uma versão anterior da busca tabu apresentada em Widmer (1991) e o método proposto obteve melhores resultados para a maior parte das instâncias.

Agnētis et al. (1997) também utilizaram busca tabu com o objetivo de minimizar o *makespan*, porém, em um ambiente industrial diferente. O SMF abordado possui duas máquinas flexíveis paralelas sem *magazine* e com compartilhamento de ferramentas entre si. A solução inicial foi gerada a partir do ordenamento decrescente das tarefas pelo tempo de processamento utilizando-se a rotina *Longest Processing Time* (LPT) e posterior distribuição das tarefas entre as máquinas, sempre atribuindo-se a próxima tarefa à máquina com menor tempo de processamento total até aquele momento.

Neste último trabalho, a partir da solução inicial são realizadas perturbações na solução buscando escapar de ótimos locais. Sempre que uma ferramenta é solicitada ao mesmo tempo por duas tarefas alocadas em máquinas diferentes, a ferramenta é utilizada

primeiramente pela tarefa com menor tempo de processamento. Os resultados não foram comparados a outros trabalhos acadêmicos. Embora o ambiente estudado seja bastante específico, ele pode ser abstraído em uma instância especial do IPMTC na qual o *magazine* das máquinas possui tamanho igual a 1 e é acrescentada às ferramentas uma limitação referente a disponibilidade das mesmas.

Mohamed, Kumar e Motwani (1999) propõem dois modelos de programação linear inteira para os subproblemas de agrupamento e roteamento de tarefas, buscando minimizar o *makespan*. O modelo denominado *Loading Model* não considera as limitações impostas pela capacidade limitada do *magazine* das máquinas e, portanto, não precisa considerar o agrupamento de tarefas. O resultado deste modelo é utilizado como limitante inferior para o valor da solução. O segundo modelo desenvolvido, denominado *Part Grouping, Loading and Routing Model* é comparado a outros modelos disponíveis na literatura e apresenta menores valores de *makespan* e uma maior flexibilidade no roteamento das tarefas. Entretanto, é importante ressaltar que o objetivo dos modelos utilizados na comparação não era a minimização do *makespan*, o que torna a comparação menos justa.

Motivados por um caso real de uma grande produtora de motores à diesel para navios e usinas de energia elétrica, Persi et al. (1999), com o objetivo de fornecer uma estratégia de planejamento de produção que melhorasse a utilização das máquinas, propõem uma abordagem hierárquica composta por quatro subníveis.

No nível superior, o problema de agrupamento das tarefas é resolvido por um modelo de programação linear inteira que minimiza o tempo de processamento dos lotes e provê um melhor balanceamento de carga entre as máquinas. Os níveis intermediários correspondentes ao sequenciamento e transição entre os lotes são modelados como uma instância do problema do Caixeiro Viajante (ou *Traveling Salesman Problem*, TSP) e resolvidos com métodos heurísticos encontrados na literatura. Para o nível inferior, o sequenciamento de tarefas dentro dos lotes, são apresentadas 10 abordagens diferentes. A abordagem *Earliest Due Date* obteve os melhores resultados, entretanto, como todas as tarefas possuíam o mesmo prazo (ou *deadline*), prazos artificiais tiveram que ser adicionadas a cada tarefa. Após a resolução de cada subproblema foi possível calcular o resultado em termos de utilização global das máquinas, atingindo a taxa de 68% de utilização, o que aproxima-se bastante da utilização crítica indicada de 74%.

Todos os trabalhos apresentados até aqui consideraram versões *offline* do IPMTC. Nesta versão, todas as tarefas são conhecidas *a priori* e estão disponíveis para serem escalonadas no tempo zero. Kurz e Askin (2001) apresentam um trabalho que aborda também a versão *online* do IPMTC. Nesta versão, as tarefas estão disponíveis para serem alocadas nas máquinas somente após um tempo variável r , sendo $r \geq 0$. Com o objetivo de minimizar o *makespan* são apresentadas quatro heurísticas baseadas no TSP, sendo duas para cada versão do problema, *offline* e *online*.

As duas primeiras heurística resolvem o problema *offline* do IPMTC. Especificamente, são utilizados o método *Doubling* para o TSP e uma versão adaptada do método MULTI-FIT. A terceira heurística faz uma adaptação da heurística de inserção para o TSP. A quarta heurística consiste em um algoritmo genético em que as mutações são feitas trocando-se as tarefas entre as máquinas e as soluções são posteriormente geradas pelo método *Doubling* para o TSP. Os resultados apresentados pelas heurísticas são comparados apenas entre si.

Como mencionado, a abordagem do IPMTC (PERSI et al., 1999; KURZ; ASKIN, 2001; FATHI; BARNETTE, 2002) e também do MTSP como uma instância do TSP é recorrente na literatura. Entretanto, um estudo apresentado por Azevedo e Carvalho (2017) demonstrou que tal modelagem não é adequada para as instâncias atuais do problema, dada a dificuldade de expressar o número exato de trocas de ferramentas na modelagem.

Fathi e Barnette (2002) abordam o IPMTC considerando um ambiente que é utilizado como definição padrão para as abordagens atuais do problema, conforme descrito em detalhes no Capítulo 3 desta dissertação. São apresentadas diferentes heurísticas com o objetivo de minimizar o *makespan*. Estas abordagens baseiam em métodos de processamento de listas de tarefas, métodos de busca local combinados e modelagem via TSP.

O clássico método de processamento de listas de tarefas consiste em criar uma lista ordenada de tarefas e distribuí-las entre as máquinas. Em outras palavras, as tarefas a serem processadas são alocadas em uma lista e sempre que uma máquina se tornar disponível, a primeira tarefa na lista é atribuída a ela. Dois diferentes tipos de listas foram considerados. O primeiro tipo de lista é gerado aleatoriamente e utiliza a heurística *Multiple-Start Greedy* (MSG) apresentada por Crama et al. (1994), para obter o sequenciamento das tarefas em cada máquina e assim calcular o *makespan*. Este método, chamado de *Random List Processing Procedure* (RLPP), gera 50 listas aleatoriamente e as sequencia via MSG, selecionando ao final a solução com melhor valor de *makespan*. O segundo tipo de lista utiliza a conhecida heurística LPT, descrita anteriormente. Entretanto, tal heurística não considera o tempo gasto com as trocas de ferramentas.

As heurísticas baseadas em busca local partem de uma solução inicial gerada pelo método RLPP e utilizam as estruturas de vizinhança de inserção e troca de tarefas. A utilização destas estruturas isoladamente resulta no método *Local Improvement Procedure* (LIP). Os autores utilizam uma combinação de ambas, resultando no método *Composite Local Improvement Procedure* (CLIP). São apresentadas duas versões desta heurística, denominadas CLIP1 e CLIP2.

A terceira heurística foi desenvolvida considerando-se as similaridades entre o IPMTC e o k-TSP. A solução inicial é obtida à partir da solução ótima (ou subótima) considerando-se a versão do problema com apenas uma máquina. Esta solução é dividida

em m subsequências dividindo-se a sequência original em $m - 1$ pontos. Cada subsequência é resolvida como uma instância do k-TSP utilizando-se um método construtivo não especificado pelos autores. O método desenvolvido foi denominado *Constructive Heuristic* (CSTR).

Para averiguar-se a qualidade das soluções, foram implementados dois limitantes inferiores baseados na premissa que a carga de trabalho seria igualmente dividida entre todas as máquinas, assim $LB_1 = \max\{0, L - mC\}$ e $LB_2 = \max\{0, L' - (m - 1)C\}$, em que L é o número total de ferramentas, m o número de máquinas, C a capacidade do *magazine* das máquinas e L' o número mínimo de trocas de ferramentas necessárias considerando-se uma solução ótima para apenas uma máquina. Utilizando-se LB_1 e LB_2 , foram definidos os limitantes inferiores para o *makespan* $MLB_1 = (T_{sum} + h \times LB_1)$ e $MLB_2 = (T_{sum} + h \times LB_2)$, onde T_{sum} é a soma do tempo de processamento total de todas as tarefas e h o tempo necessário para a troca de uma ferramenta.

Os experimentos computacionais compararam os cinco procedimentos desenvolvidos, CLIP1, CLIP2, CSTR, RLPP e LPT, em três grupos de instâncias. Considerando-se a totalidade das instâncias, CLIP2 e CLIP1 obtiveram os melhores resultados de *makespan*. Embora CSTR tenha obtido o melhor valor de *makespan* em algumas instâncias, seu resultado médio final foi 50% maior do que o obtido por CLIP2. Os resultados apresentados por LPT são os piores entre os cinco métodos implementados, o que era esperado pois o mesmo não considera o tempo necessário para a troca de ferramentas. Os limitantes inferiores desenvolvidos mostraram-se ineficazes, pois apresentaram valores de *makespan* superiores ao obtido por CLIP1 e CLIP2 em uma instância.

O trabalho de [Fathi e Barnette \(2002\)](#) permaneceu como estado da arte por mais de uma década. Durante este período, nenhum trabalho abordando o IPMTC sem variações foi publicado. Recentemente, [Beezão et al. \(2017\)](#) retomaram o tema propondo uma implementação da metaheurística *Adaptive Large Neighborhood Search* (ALNS), proposta originalmente por [Ropke e Pisinger \(2006\)](#), e dois modelos de programação linear inteira. O ALNS, partindo de uma solução inicial, utiliza um conjunto de heurísticas de inserções e remoções no intuito de explorar de forma eficiente um grande número de vizinhanças da solução atual à procura de soluções melhores.

Os autores utilizam duas estratégias para gerar as soluções iniciais. A primeira utiliza uma heurística construtiva desenvolvida por [Chaves, Senne e Yanasse \(2012\)](#) para o MTSP aliada a um modelo de programação linear inteira proposto no próprio trabalho, para transformar a solução do MTSP em uma solução para o IPMTC. A esta abordagem denominou-se ALNS-B. A segunda estratégia gera as soluções iniciais aleatoriamente, a qual denominou-se ALNS-R. Foram elaborados nove operadores de remoção e cinco operadores de inserção considerando-se características tanto do PM quanto do IPMTC. Além do modelo mencionado, duas formulações matemáticas para a resolução do IPMTC

foram propostas. O primeiro modelo, denominado M_1 é inspirado no trabalho de [Tang e Denardo \(1988\)](#) para o MTSP e o segundo, denominado M_2 , é inspirado no trabalho de [Laporte, Salazar-Gonzalez e Semet \(2004\)](#) também para o IPMTC.

Os experimentos computacionais consideraram dois conjuntos de 1440 instâncias, denominados IPMTC-I e IPMTC-II, gerados neste mesmo trabalho. O conjunto IPMTC-I é adaptado de [Yanasse \(2009\)](#) e contém as menores instâncias. Para instâncias com 8 tarefas o número de máquinas é igual a 2. Para instâncias com 15 tarefas, o número de máquinas varia entre 2 e 3. Para instâncias com 25 tarefas, o número de máquinas varia de 2 a 4. Para o conjunto IPMTC-II foram geradas instâncias maiores. Para instâncias com 50 tarefas, o número de máquinas varia entre 3 e 5. Para instâncias com 100 tarefas, o número de máquinas varia entre 4 e 7. Para instâncias com 200 tarefas, o número de máquinas varia entre 6 e 10.

As formulações M_1 e M_2 foram avaliadas considerando um subconjunto contendo 30 instâncias pertencentes ao conjunto IPMTC-I. Para as instâncias com 8 tarefas, ambas formulações alcançaram 100% de resultados ótimos. Entretanto, para instâncias maiores os modelos encontraram dificuldades. Para instâncias com 15 tarefas nenhum resultado ótimo foi alcançado e para instâncias com 25 tarefas apenas 10% de resultados ótimos foram obtidos. De acordo com os autores, o modelo M_1 obteve resultados de melhor qualidade no geral. A Seção 3.1.1 apresenta os detalhes deste modelo.

As duas versões do método ALNS foram comparadas entre si e com os métodos CLIP1 e CLIP2 de [Fathi e Barnette \(2002\)](#), em uma nova implementação realizada pelos autores, embora detalhes não tenham sido apresentados. Os valores das soluções obtidas pelos métodos comparados não são reportados pelos autores, apenas o *gap* e o tempo de processamento são relatados. Segundo os dados apresentados, as duas versões do ALNS apresentaram resultados melhores que CLIP1 e CLIP2 em todas as instâncias do conjunto IPMTC-I. A qualidade das soluções apresentadas pelas versões ALNS-B e ALNS-R são comparáveis em qualidade, com ALNS-R apresentando um *gap* médio um pouco menor. Para as instâncias do conjunto IPMTC-II, o método ALNS-B apresentou dificuldades já nas instâncias com 50 tarefas. Para as instâncias com 100 e 200 tarefas o método atingiu o tempo limite de 3600 segundos antes mesmo de determinar a solução inicial. O método ALNS-R apresentou resultados melhores que CLIP1 e CLIP2 em todas as instâncias deste conjunto.

[Gökgür, Hnich e Özpeynirci \(2018\)](#) abordam o escalonamento de tarefas em um ambiente com máquinas paralelas não relacionadas. Com objetivo minimizar o *makespan* são apresentados dois modelos de programação linear inteira mista e uma implementação de busca tabu. Entretanto, o ambiente proposto não considera o tempo de troca de ferramentas e acrescenta um número limitado de cópias para cada ferramenta.

Conforme relatado, o método proposto por [Beezão et al. \(2017\)](#) apresentou resulta-

dos melhores que CLIP1 e CLIP2 em todas as 2880 instâncias do conjunto de instâncias *benchmark* proposto. Assim, o método ALNS é o estado atual da arte para o IPMTC. Nos experimentos computacionais relatados no Capítulo 5 são considerados as instâncias e os métodos que compõem o estado da arte atual do IPMTC.

3 Fundamentação teórica

Este capítulo apresenta a fundamentação teórica a respeito do Problema de Escalonamento de Tarefas em Máquinas Idênticas Paralelas com Limitações de Ferramentas e a respeito do método utilizado neste trabalho para sua abordagem. Nas seções seguintes são apresentadas uma definição formal do IPMTC e a conceituação do Algoritmo Genético de Chaves Aleatórias Viciadas.

3.1 O problema de escalonamento de tarefas em máquinas idênticas paralelas com limitações de ferramentas

Dado um SMF contendo um conjunto de máquinas flexíveis idênticas paralelas $M = \{1, \dots, m\}$, com capacidade para comportar até C ferramentas simultaneamente em seu *magazine*. Um conjunto de tarefas a serem processadas $T = \{1, \dots, n\}$. Um conjunto de ferramentas $L = \{1, \dots, l\}$ e um subconjunto de ferramentas L_i ($i \in L$) necessárias para processar a tarefa i ($i \in T$), sendo $|L_i| \leq C$, um tempo de processamento para cada tarefa indicado por t_i ($i \in T$) e um tempo constante \bar{t} para troca de cada ferramenta. O IPMTC consiste em determinar a alocação e o sequenciamento das tarefas nas máquinas k ($\forall k \in M$) de forma que este sequenciamento resulte no menor número possível de trocas de ferramentas e, conseqüentemente, no menor tempo de produção visando a minimização do *makespan*, representado por Δ .

A definição formal do IPMTC foi apresentada por [Fathi e Barnette \(2002\)](#) de acordo com as seguintes características:

- Todas as tarefas são conhecidas e estão prontas para serem processadas desde o início;
- Não há ordem de precedência entre as tarefas;
- O processamento de uma tarefa não pode ser interrompido antes do seu final;
- Cada máquina possui um conjunto completo de ferramentas;
- Todas as máquinas possuem um *magazine* com capacidade limitada, porém, grande o suficiente para o processamento de qualquer tarefa individualmente;
- O tempo necessário para trocar uma ferramenta é o mesmo para todas as ferramentas;
- As ferramentas podem ocupar qualquer posição disponível no *magazine*;

- O tempo necessário para a configuração inicial da máquina não é considerado.

Considerando-se as características citadas e o sistema de classificação proposto por [Graham et al. \(1979\)](#), o IPMTC pertence à classe $Pm|s_{jk}|C_{max}$, em que Pm representa a disposição em paralelo do conjunto de máquinas idênticas, s_{jk} os tempos de preparação dependentes da sequência de processamento das tarefas e C_{max} representa o *makespan*. Especificamente para o IPMTC, o valor s_{jk} indica um tempo de preparo dependente de todo o sequenciamento de tarefas e não apenas das tarefas j e k . Visto que o *magazine* das máquinas pode possuir capacidade superior à quantidade de ferramentas utilizadas por uma tarefa, ferramentas ainda não utilizadas podem ser previamente carregadas a fim de evitar trocas desnecessárias. De forma análoga, uma ferramenta não utilizada pela tarefa atual, havendo disponibilidade de espaço no *magazine*, pode ser mantida se for necessária por uma tarefa posterior. Sendo assim, todo o conjunto de tarefas deve ser considerado no tempo de preparo da máquina.

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 2 apresenta um modelo de instância válida em que se considera $C = 5$, $m = 2$ e $\bar{t} = 1$.

Tabela 2 – Exemplo de instância do IPMTC.

Tarefas	Tempo de Processamento	Ferramentas
1	10	1, 2, 3
2	10	1, 2, 4
3	12	1, 2
4	12	5, 6, 7
5	5	6, 8, 9

O subconjunto de ferramentas necessário para o processamento de uma determinada tarefa pode ser representado por uma matriz binária Q . Cada linha da matriz corresponde a uma tarefa e cada coluna corresponde a uma ferramenta. Se uma tarefa i precisa da ferramenta j , então $q_{ij} = 1$. Caso contrário $q_{ij} = 0$. A matriz Q para o exemplo de instância mostrado na Tabela 2 pode ser vista na Tabela 3.

Tabela 3 – Exemplo de matriz de ferramentas.

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

Conforme demonstrado por [Fathi e Barnette \(2002\)](#), a solução do IPMTC passa obrigatoriamente por três etapas:

1. Alocação das tarefas às máquinas;
2. Sequenciamento das tarefas em cada máquina;
3. Determinação do plano de troca de ferramentas para a sequência dada em cada máquina.

Após a alocação das tarefas às máquinas, o problema se reduz a uma instância do MTSP. Definida a sequência de processamento das tarefas, o plano de trocas de ferramentas pode ser resolvido em tempo determinístico polinomial utilizando-se o 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 1 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.

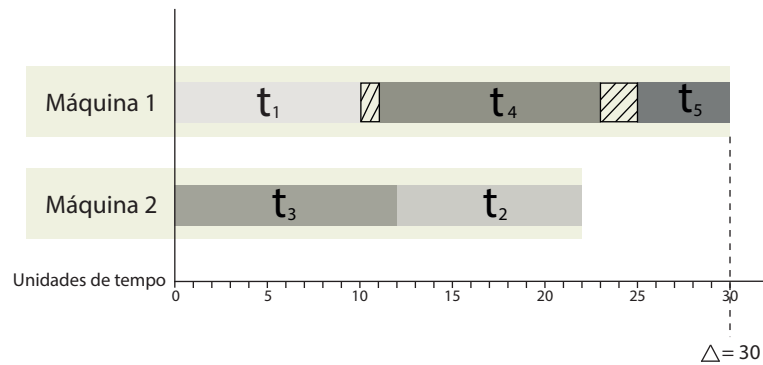


Figura 1 – Exemplo de solução para a instância apresentada na Tabela 2.

O tempo de processamento total de cada máquina, indicado por Δ_m ($m \in M$), é calculado pela soma dos tempos de processamento das tarefas e o tempo gasto com as trocas de ferramentas. No exemplo mostrado na Figura 1, o tempo de processamento total da máquina 1 é a soma do tempo de processamento das tarefas a ela atribuídas [10, 12, 5], com o tempo gasto para as trocas de ferramenta, neste exemplo, considera-se que o *magazine* da máquina 1 foi inicialmente preenchido com as ferramentas {1, 2, 3, 5 e 6}. Assim, foi necessário uma troca de ferramenta antes do processamento tarefa 4 e mais duas trocas antes do processamento da tarefa 5. O tempo de processamento total da máquina 1 é $10 + 12 + 5 + 1 + 2 = 30$.

Para a máquina 2 são alocadas duas tarefas com os tempos de processamento [12, 10]. O *magazine* da máquina 2 comporta todas as ferramentas necessárias para o processamento destas duas tarefas {1, 2 e 4} e elas são carregadas em sua configuração inicial. Como não há trocas de ferramentas, o tempo total é igual a soma do tempo de processamento das tarefas, ou seja, $12 + 10 = 22$. O *makespan* é o maior tempo de processamento total entre as máquinas, neste exemplo, o tempo de processamento total da máquina 1, correspondente a 30 unidades de tempo.

Embora a solução apresentada pela Figura 1 seja válida, esta não é ótima. A Figura 2 apresenta uma solução ótima para a mesma instância. São alocadas à máquina 1 as tarefas [1, 2, 5] com os respectivos tempos de processamento [10, 10, 5]. O *magazine* da máquina 1 é inicialmente preenchido com as ferramentas {1, 2, 3, 4 e 6} e duas trocas de ferramentas são necessárias antes que a tarefa 5 seja processada. Assim, o tempo de processamento total da máquina 1 é $10 + 10 + 5 + 2 = 27$. Para a máquina 2 são alocadas as tarefas [3, 4] com tempos de processamento [12, 12]. O *magazine* da máquina 2 comporta todas as ferramentas necessárias para o processamento destas duas tarefas {1, 2, 5, 6 e 7} e elas são carregadas em sua configuração inicial. Como não há trocas de ferramentas, o tempo total é igual a soma do tempo de processamento das tarefas, ou seja, $12 + 12 = 24$. O *makespan* para esta solução é 27 unidades de tempo.

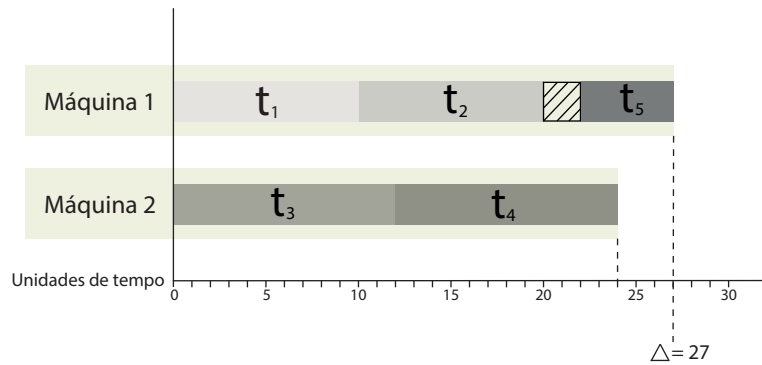


Figura 2 – Solução ótima para a instância apresentada na Tabela 2.

O plano de troca de ferramentas de cada máquina pode ser representado por uma matriz binária $R^m = \{r_{ij}^m\}$, com $i \in \{1, \dots, l\}, j \in \{1, \dots, n\}$. Em R^m , as linhas representam as ferramentas disponíveis e as colunas representam cada tarefa alocada à máquina m , em ordem de processamento. Um elemento $r_{ij}^m = 1$ indica que a ferramenta i está carregada na máquina m durante o processamento da tarefa j , e $r_{ij}^m = 0$ caso contrário.

A Tabela 4 apresenta os planos de trocas referentes à solução anterior. 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.

Tabela 4 – Exemplo do plano de troca de ferramentas.

(a) Máquina 1				(b) Máquina 2		
Ferramentas	1	2	5	Ferramentas	3	4
1	1	1	<u>0</u>	1	1	1
2	1	1	<u>0</u>	2	1	1
3	1	1	1	3	0	0
4	1	1	1	4	0	0
5	0	0	0	5	1	1
6	1	1	1	6	1	1
7	0	0	0	7	1	1
8	0	0	<u>1</u>	8	0	0
9	0	0	<u>1</u>	9	0	0

Uma solução representada pela matriz R^m é avaliada pela Equação (3.1) 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. Esta função é utilizada como função objetivo do MTSP.

$$Z_{MTSP}(R^m) = \sum_{i \in T} \sum_{j \in L} r_{ji}^m (1 - r_{ji-1}^m) \quad (3.1)$$

A função objetivo do IPMTC é a minimização do *makespan* (Δ), resultado da Equação (3.2). A função realiza a soma entre o tempo de processamento da tarefa (t_i) e a multiplicação entre o número de trocas de ferramentas imediatamente anterior ao processamento da tarefa corrente ($Z_{MTSP}(R^m)$) e o tempo necessário para a troca de uma ferramenta (\bar{t}).

$$\Delta = \arg \max \left\{ \sum_{i \in m} t_i + Z_{MTSP}(R^m) \times \bar{t} \right\}, \forall m \in M \quad (3.2)$$

3.1.1 Formulação matemática de [Beezão et al. \(2017\)](#)

A formulação matemática M_1 proposta por [Beezão et al. \(2017\)](#) é a de melhor desempenho na literatura. Inspirado pelo trabalho de [Tang e Denardo \(1988\)](#) para o MTSP, o modelo utiliza três famílias de variáveis binárias:

- w_{jkm} , igual a 1 se, e somente se, a tarefa j é processada na máquina m na posição k ;
- y_{tkm} , igual a 1 se, e somente se, a ferramenta t está alocada na máquina m durante o processamento da k -ésima tarefa;

- z_{tkm} , igual a 1 se, e somente se, a ferramenta t é inserida no *magazine* da máquina m exatamente antes do início do processamento da tarefa na posição k .

Sendo J_t o conjunto de tarefas que requerem a ferramenta t , P_j o tempo de processamento da tarefa j , Δ_m o tempo de processamento da máquina m , e Δ o *makespan*, o modelo M_1 é dado por:

$$\text{Minimizar} \quad \Delta \quad (3.3)$$

$$\text{Sujeito a} \quad \sum_{j=1}^n w_{jkm} \leq 1 \quad k \in J, m \in M \quad (3.4)$$

$$\sum_{m \in M} \sum_{k=1}^n w_{jkm} = 1 \quad j \in J \quad (3.5)$$

$$\sum_{j=1}^n w_{jkm} \leq \sum_{j=1}^n w_{j(k-1)m} \quad k \in J \setminus \{1\}, m \in M \quad (3.6)$$

$$\sum_{j \in J_t} w_{jkm} \leq y_{tkm} \quad t \in T, k \in J, m \in M \quad (3.7)$$

$$\sum_{t=1}^l y_{tkm} \leq C \quad k \in J, m \in M \quad (3.8)$$

$$y_{tkm} - y_{t(k-1)m} \leq z_{tkm} \quad t \in T, k \in J \setminus \{1\}, m \in M \quad (3.9)$$

$$\sum_{k=1}^n \sum_{j=1}^n p_j w_{jkm} + \bar{t} \sum_{k=2}^n \sum_{t=1}^l z_{tkm} \leq \Delta_m \quad m \in M \quad (3.10)$$

$$\Delta_m \leq \Delta \quad m \in M \quad (3.11)$$

$$w_{jkm} \in \{0, 1\} \quad j \in J, k \in J, m \in M \quad (3.12)$$

$$y_{tkm} \in \{0, 1\} \quad t \in T, k \in J, m \in M \quad (3.13)$$

$$z_{tkm} \in \{0, 1\} \quad t \in T, k \in J, m \in M \quad (3.14)$$

A função objetivo (3.3) minimiza o *makespan*. As restrições de (3.4) a (3.6) referem-se a alocação das tarefas às máquinas. Em (3.4) garante-se que cada tarefa seja alocada no máximo uma vez e em apenas uma máquina. Em (3.5) obriga-se a alocação de todas as tarefas e, em (3.6), garante-se que uma tarefa só possa ser executada na posição k de uma máquina m após a execução da tarefa na posição $k - 1$ na mesma máquina m .

As restrições de (3.7) a (3.9) controlam a utilização das ferramentas. Em (3.7) garante-se que todas as ferramentas necessárias ao processamento de uma tarefa j estejam presentes no *magazine* da máquina no momento de sua execução. Em (3.8) certifica-se que a capacidade do *magazine* não seja ultrapassada e, em (3.9), controla-se a inserção de ferramentas no *magazine*.

A restrição (3.10) calcula o tempo de execução total de cada máquina e o *makespan* é definido em (3.11). As restrições de (3.12) a (3.14) garantem a integralidade das variáveis de decisão.

O Capítulo 5 apresenta um experimento computacional em que o modelo M_1 é utilizado para gerar soluções ótimas e limitantes inferiores para as instâncias consideradas nesta dissertação.

3.2 Algoritmo genético de chaves aleatórias viciadas

Problemas \mathcal{NP} -Difíceis não possuem algoritmos determinísticos polinomiais conhecidos para solucioná-los. Uma dificuldade recorrente quando aborda-se tais problemas é escapar de ótimos locais. Um ótimo local ocorre quando todas as soluções vizinhas de uma determinada solução são piores do que a solução já encontrada. Não há garantias que um ótimo local seja também a melhor solução possível para o problema, o ótimo global. Assim, faz-se necessário fugir desta região para outra, onde a vizinhança ainda não explorada pode conter ótimos locais melhores ou mesmo um ótimo global. Neste contexto, diversos métodos foram desenvolvidos apresentando estratégias genéricas para coordenar métodos de intensificação e diversificação da busca por soluções e também escapar de ótimo locais, aumentando as possibilidades de encontrar um ótimo global ou uma solução mais próxima deste. Esses métodos são classificados como metaheurísticas e podem facilmente ser adaptados para resolver diversos problemas.

Algumas metaheurísticas foram inspiradas na teoria da evolução de *Charles Darwin*, sendo por isso conhecidas como Algoritmos Evolutivos. Estas metaheurísticas trabalham evoluindo uma população de soluções 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 irão 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 e genes para representar as soluções que são tratadas como indivíduos de uma população. Cada indivíduo é representado por um cromossomo, que por sua vez é composto por uma cadeia de valores chamados genes. As novas gerações são criadas através da recombinação (*crossover*) dos cromossomos das gerações anteriores. Usualmente, dois cromossomos da geração atual, chamados de pais, são combinados em um ou mais cromossomos descendentes que serão parte da população da geração seguinte. Para diversificar a população e diminuir a possibilidade 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.

Bean (1994) apresentou uma nova classe de algoritmos genéticos denominada Algoritmos Genéticos de Chaves Aleatórias (*Random-Key Genetic Algorithms*, RKGA). Como em algoritmos genéticos tradicionais, os *cromossomos* são representações de soluções, entretanto, a *codificação* utilizada é a de vetores cujos conteúdos, também denominados *genes*, são chaves geradas aleatoriamente no intervalo contínuo $[0,1]$. Um exemplo de cromossomo com seis chaves aleatórias é mostrado na Figura 3.

0,3256	0,9863	0,0007	0,1258	0,1456	0,9971
--------	--------	--------	--------	--------	--------

Figura 3 – Exemplo de cromossomo com seis chaves aleatórias.

Uma *população* P no RKGA é representada por um conjunto de p cromossomos possuindo x chaves aleatórias cada, conforme apresentado na Figura 4, em que o *fitness* é apresentado no círculo próximo a cada cromossomo. A população *evolui* aplicando-se o princípio *Darwinista* no qual os melhores indivíduos de uma população têm maiores chances de se reproduzir, e com isso, perpetuar seu material genético nas próximas gerações. Para este fim são utilizados operadores de *seleção*, *reprodução* e *mutação*.

0,4128	0,7865	0,8765	0,2882	0,1245	0,8967	13
0,5678	0,5432	0,2332	0,6754	0,6557	0,7342	17
0,7865	0,1093	0,9803	0,0987	0,1234	0,8765	14
0,6754	0,3221	0,9087	0,6754	0,7865	0,8976	21
0,6754	0,2345	0,6547	0,7689	0,9874	0,0897	10
0,5467	0,4678	0,8907	0,5467	0,8734	0,8329	08

Figura 4 – Exemplo de população de cromossomos.

O RKGA parte da população inicial P , gerada de maneira totalmente aleatória. O processo evolutivo é dividido em *gerações* consecutivas, cada uma com sua própria população gerada com base na aplicação dos operadores evolutivos na população anterior. Na k -ésima geração, a população é separada em dois conjuntos denominados *elite* e *não-elite*. O conjunto elite, ou p_e , corresponde as melhores soluções encontradas de acordo com

o *fitness*, ou adaptação, calculado pela função objetivo e nunca é maior que a metade da população (i.e., $p_e < p/2$). O conjunto não-elite corresponde ao restante da população. Todos os cromossomos do conjunto elite são copiados sem alteração para a população da $k + 1$ -ésima geração. A Figura 5 ilustra um exemplo desta classificação dos cromossomos considerando um hipotético problema de maximização.

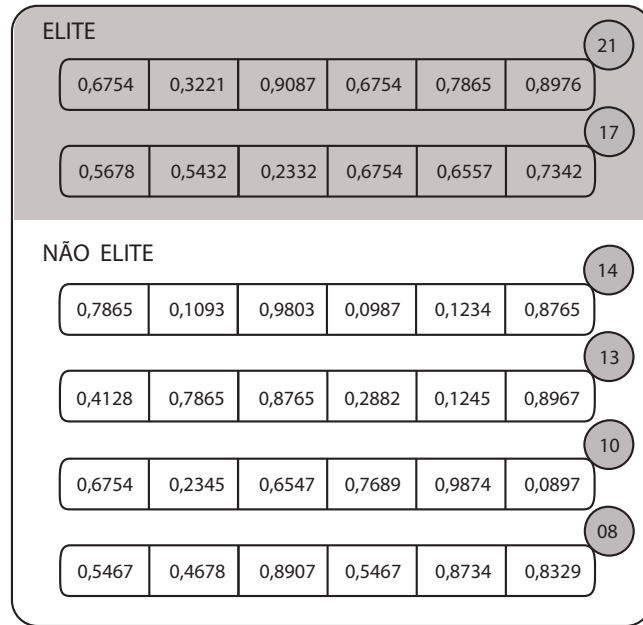


Figura 5 – Exemplo de classificação da população em elite e não-elite.

Em seguida, são gerados e inseridos na população da $k + 1$ -ésima geração p_m cromossomos denominados *mutantes*. Assim como nos algoritmos genéticos clássicos, o operador de mutação tem como função evitar a convergência prematura para um ótimo local não global, por meio da inserção de novos cromossomos gerados de maneira aleatória ou cromossomos alterados aleatoriamente. Dada a intensificação gerada pelo elitismo, a mutação no RKGA consiste em gerar novos cromossomos de maneira aleatória.

O restante da população da $k + 1$ -ésima geração é formado pela reprodução (ou *crossover*) de pares de cromossomos da k -ésima geração escolhidos aleatoriamente. É utilizada 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 *pai* com a mesma probabilidade. O operador de reprodução tem como função intensificar a busca por soluções. A Figura 6 apresenta a construção de um cromossomo filho a partir do cruzamento dos genes de dois cromossomos pais.

O processo de *decodificação* de um cromossomo para a obtenção da solução real é dependente do problema tratado e requer conhecimento específico. O processo evolutivo se repete até que algum critério de parada seja satisfeito, como número máximo de gerações, tempo decorrido e critérios relativos à qualidade das soluções obtidas, entre outros.

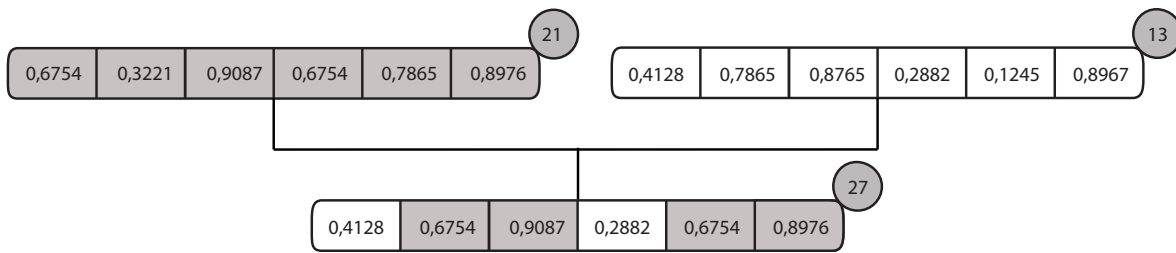


Figura 6 – Exemplo de aplicação do operador de reprodução.

Inspirado no RKGA, [Resende \(2011\)](#) apresenta o Algoritmo Genético de Chaves Aleatórias Viciadas (*Biased Random-key Genetic Algorithm*, BRKGA). O BRKGA difere do RKGA na forma como os pais são selecionados e como o cruzamento é implementado. Enquanto no RKGA ambos os pais são escolhidos da população inteira, no BRKGA um dos pais é sempre escolhido no conjunto elite enquanto o outro é escolhido do conjunto não-elite.

Assim como no RKGA, o BRKGA realiza a reprodução de dois cromossomos pais utilizando a combinação uniforme parametrizada de [Spears e Jong \(1995\)](#). Entretanto, no BRKGA, 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 quase sempre obtém melhores soluções que o RKGA. O funcionamento geral do BRKGA é ilustrado pela Figura 7.

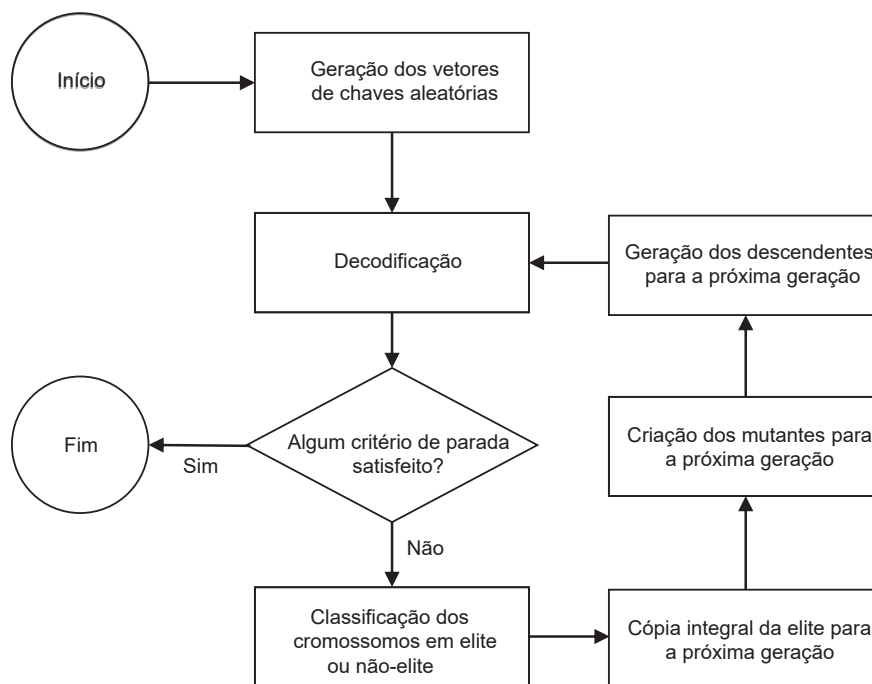


Figura 7 – Visão geral do funcionamento do BRKGA. Adaptado de [Toso e Resende \(2015\)](#).

Neste trabalho, o BRKGA é utilizado para abordagem do IPMTC. Os detalhes da implementação, bem como a hibridização deste método com buscas locais são descritos no Capítulo [4](#).

4 Desenvolvimento

Embora o IPMTC seja um problema bem definido e estudado por um longo período, não há registros na literatura de sua abordagem utilizando-se o BRKGA. Para que seja possível implementar tal método, as etapas de codificação e decodificação do BRKGA necessitam ser adequadas conforme o problema abordado. A seguir, são detalhadas as modificações realizadas ao BRKGA padrão para abordagem do IPMTC. Adicionalmente, optou-se por hibridizar o BRKGA com a utilização de buscas locais, o que também é descrito a seguir.

4.1 Codificação

Como mencionado, cada cromossomo da população em um BRKGA é representado por um vetor de x chaves aleatórias no intervalo contínuo $[0, 1] \in \mathbb{R}$. Para retratar fielmente uma solução do IPMTC, 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. A quantidade de chaves em cada cromossomo é definida pelo número n de tarefas. O princípio desta codificação é que a parte inteira da chave aleatória indique em qual máquina i cada tarefa será alocada. A parte decimal da chave será útil na decodificação ao determinar a posição de uma tarefa específica no sequenciamento da máquina i . A Figura 8 apresenta a codificação de um cromossomo válido para uma instância IPMTC contendo duas máquinas e seis tarefas.

1,3256	2,9863	2,0007	1,1258	2,1456	2,9971
--------	--------	--------	--------	--------	--------

Figura 8 – Codificação de um cromossomo para uma instância do IPMTC com duas máquinas e seis tarefas.

4.2 Decodificação

O decodificador é descrito por [Gonçalves, Resende e Toso \(2012\)](#) como um algoritmo determinístico que, tomando como entrada um vetor de n chaves aleatórias, produz como saída uma solução do problema de otimização assim como o custo desta solução. Para decodificar o vetor de chaves aleatórias em uma solução IPMTC válida, é necessário ordenar o vetor de forma não decrescente de acordo com o valor das chaves, mantendo junto com a chave o seu índice original. Por exemplo, O cromossomo mostrado como exemplo na Figura 8 é ordenado conforme mostrado na Figura 9.

Índice original	3	0	2	4	1	5
Genes	1,1258	1,3256	2,0007	2,4156	2,9863	2,9971

Figura 9 – Cromossomo com chaves ordenadas, em parte do processo de decodificação.

Conforme mencionado, a parte inteira de cada chave 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 vetor ordenado, ou seja, por sua parte decimal. Assim, o exemplo mostrado pela Figura 9 corresponde à alocação das tarefas 3 e 0 à máquina 1 e alocação das tarefas 2, 4, 1 e 5 à máquina 2, nesta ordem.

O tempo de processamento total da máquina i é calculado somando-se o tempo de processamento de todas as tarefas a ela alocadas com a multiplicação do tempo necessário para a troca de uma ferramenta pelo número de trocas necessárias. O número de trocas necessárias é obtido utilizando-se a política KTNS de Tang e Denardo (1988). 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 IPMTC é minimizar o *makespan*, a solução para a instância será o cromossomo com menor valor de *fitness* ao final do processo evolutivo.

4.3 Funções de avaliação

Conforme mencionado no Capítulo 3, a função objetivo do IPMTC é a minimização do *makespan*, resultado da Equação (3.2). Durante a execução do BRKGA, duas funções de avaliação são utilizadas para aferir a qualidade das soluções, uma que avalia completamente uma dada solução e outra que avalia somente o efeito pontual de uma determinada modificação em uma solução. A seguir, essas funções são descritas em detalhes.

4.3.1 Avaliação completa

Após a definição da ordem de processamento de um conjunto de tarefas em uma dada máquina, torna-se necessário determinar o seu tempo de processamento, composto pela soma dos tempos de processamento das tarefas e pelo tempo relacionado ao total de trocas de ferramentas. Conforme mencionado anteriormente, o maior tempo de processamento de uma máquina corresponde ao *makespan*.

A soma dos tempos de processamento das tarefas é trivial, sendo realizado em tempo linear no número de tarefas. O plano de troca de ferramentas pode ser calculado em tempo $O(|L||T|)$ utilizando-se a política ótima de manter no *magazine* as ferramentas que serão utilizadas mais cedo de acordo com o método guloso *Keep Tool Needed Soonest*, introduzida por Tang e Denardo (1988). Este método visa, em caso de troca, retirar do

magazine preferencialmente as ferramentas que não serão mais utilizadas. Sendo necessário retirar alguma ferramenta que será novamente utilizada, opta-se por retirar a ferramenta que será mais tardiamente reutilizada.

4.3.2 Avaliação δ

Uma vez alterada uma solução que havia sido avaliada anteriormente de maneira completa, pode ser possível avaliar o impacto isolado da alteração realizada e verificar se houve melhoria ou piora do valor da solução. Este tipo de avaliação, denominado avaliação δ deve ser mais rápida do que uma avaliação completa e é particularmente útil em métodos de busca local.

O conceito de *blocos de zeros* e *blocos de uns* no contexto do MTSP foi apresentado por [Crama et al. \(1994\)](#) como uma sequência ininterrupta de entradas consecutivas, 0 ou 1 respectivamente, em uma mesma linha de uma matriz binária. Como demonstrado no Capítulo 3, a matriz de ferramentas de uma determinada máquina é uma matriz binária. A presença de dois ou mais blocos de uns em uma mesma linha dessa matriz pode indicar que uma ferramenta foi retirada e depois recolocada no *magazine* da máquina.

[Haddadi et al. \(2015\)](#), no contexto do problema de minimização de blocos consecutivos, apresentam uma avaliação δ capaz de verificar em tempo $O(|L|)$ o impacto no número de blocos de uns ao inserir uma única coluna em uma nova posição ou ao trocar duas colunas de posição entre si. Especificamente, o número de blocos de uns criados pela inserção de uma coluna j entre duas colunas adjacentes i e k pode ser calculado pelo número de ocorrências do padrão 101 e 010 em cada linha da matriz, dado por

$$\delta(i, j, k) = \sum_{r=1}^{|L|} (B_j^r - B_i^r \times b_j^r - B_j^r \times B_k^r + B_i^r \times B_k^r).$$

Neste trabalho, a avaliação δ de [Haddadi et al. \(2015\)](#) é utilizada para determinar se houve aumento do número de blocos consecutivos em um dos métodos de busca local. Especificamente no IPMTC, a diminuição do número de blocos de uns não necessariamente resulta em melhoria do valor da solução, conforme discutido na Seção 4.5.3. Entretanto, o aumento do bloco de uns impossibilita a melhora do valor da solução. Assim, quando procura-se melhores resultados, qualquer permutação com maior quantidade de bloco de uns pode ser descartada.

4.4 Critérios de parada

Neste trabalho, os critérios de parada do BRKGA utilizados foram o número máximo de gerações e o tempo máximo de execução. Os valores utilizados para estes critérios foram definidos nos experimentos descritos na Seção 5.2.3.

4.5 Buscas locais

Um algoritmo de busca local define, para uma dada solução s , uma vizinhança composta por um conjunto de soluções com características próximas, de acordo com um movimento específico. O algoritmo executa o movimento em s , alterando um ou mais de seus elementos e gera a solução s' , denominada solução vizinha. O valor da solução s' é determinado considerando-se a função de avaliação. Se todas as soluções vizinhas de s possuírem valor igual ou pior ao valor de s , esta representa um ótimo local em relação à vizinhança adotada. No intuito de intensificar a exploração realizada pelo BRKGA, são incorporadas ao processo evolutivo três buscas locais. A aplicação destas buscas locais está organizada como uma descida em vizinhança variável. As próximas seções descrevem em detalhes as buscas locais e a organização da aplicação das mesmas.

4.5.1 Busca local por inserção de tarefas

Conforme mencionado no Capítulo 2, ao estudar o escalonamento de tarefas em máquinas, [Stecke \(1983\)](#) lista uma série de seis objetivos que caracterizam diferentes problemas de sequenciamento em sistemas de manufatura flexível. O primeiro objetivo listado é o balanceamento de tempo de processamento entre as máquinas. Inspirado neste objetivo, esta busca local tenta encontrar soluções melhores movendo tarefas alocadas à máquina com maior tempo de processamento para a máquina com o menor tempo de processamento.

Para este fim, as máquinas que compõem uma solução são ordenadas de forma não crescente pelo tempo de processamento. Uma tarefa é selecionada aleatoriamente entre as alocadas à máquina com o maior tempo de processamento. Caso a diferença entre o tempo de processamento da primeira e última máquinas seja menor do que o tempo de processamento da tarefa selecionada, o movimento é descartado, a solução inicial retornada e o algoritmo se encerra. Caso contrário, a tarefa selecionada é removida da última máquina e alocada à primeira máquina na posição que resultar no menor número de trocas de ferramentas adicionais. Se o novo tempo de processamento da primeira máquina for menor que o *makespan*, as máquinas são reordenadas, o *makespan* é atualizado e o processo se reinicia. Caso não haja melhora na solução, a solução anterior é um ótimo local para esta vizinhança e o algoritmo se encerra.

O Algoritmo 1 apresenta as etapas desta busca local. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como a solução corrente. O laço compreendido entre as linhas 2 e 18 coordena a aplicação da busca local enquanto houver melhoria. Inicialmente, não há melhoria (linha 3) e são identificadas as máquinas com maior e menor tempo de processamento (linhas 4 e 5). Uma tarefa é aleatoriamente selecionada da máquina com maior tempo de processamento (linha 6) e é verificado se o tempo de

processamento desta tarefa somado ao tempo de processamento da máquina destino é maior do que o *makespan* atual (linha 7). Caso positivo, o movimento acarreta em piora da solução e, portanto, não é executado, encerrando a busca local (linha 8). Caso negativo, a tarefa é removida da máquina com o maior tempo de processamento e inserida na máquina de destino na posição em que acrescentar o mínimo possível de trocas de ferramentas (linhas 10 e 11). Se o novo tempo de processamento da máquina que recebeu a tarefa for menor que o *makespan*, a melhoria é registrada, a solução atualizada (linhas 13 e 14) e a busca local é reiniciada. Não havendo melhora do *makespan*, s é um ótimo local para a vizinhança explorada e a busca local termina.

Algoritmo 1: Busca local por inserção de tarefas.

Entrada: Solução s
Saída: Solução s

```

1  início
2      repita
3          melhoria ← falso;
4           $m_c \leftarrow \arg \max_{m \in [1, \dots, |M|]} \Delta_m$ ;
5           $m_f \leftarrow \arg \min_{m \in [1, \dots, |M|]} \Delta_m$ ;
6           $j \leftarrow \text{tarefa aleatória de } m_c$ ;
7          se  $t_j + \Delta_{m_f} > \Delta$  então
8              retorna  $s$ ;
9          senão
10             retire  $j$  de  $m_c$ ;
11             insira  $j$  em  $m_f$  na melhor posição;
12             se  $\Delta_{m_f} < \Delta$  então
13                 melhoria ← verdadeiro;
14                 atualize  $s$ ;
15             fim
16         fim
17     até melhoria = falso;
18     retorna  $s$ ;
19 fim

```

A Figura 10 ilustra um movimento de inserção hipotético desta busca local. Em (a) tem-se uma dada solução viável composta por duas máquinas. A máquina 01 possui o maior tempo de processamento. A seleção aleatória de uma tarefa dentre as alocadas à máquina 01 é demonstrada em (b). Neste exemplo, a tarefa 3 foi selecionada. Em seguida, conforme demonstrado em (c), a busca local procura a melhor posição para inserção da tarefa selecionada na máquina 02. Todas as posições possíveis na máquina 02 são avaliadas e a tarefa 3 é inserida na posição que acrescenta o menor número de trocas de ferramentas. Neste caso, a tarefa foi alocada à segunda posição. Em (d), temos a solução vizinha gerada pelo movimento e sua avaliação.

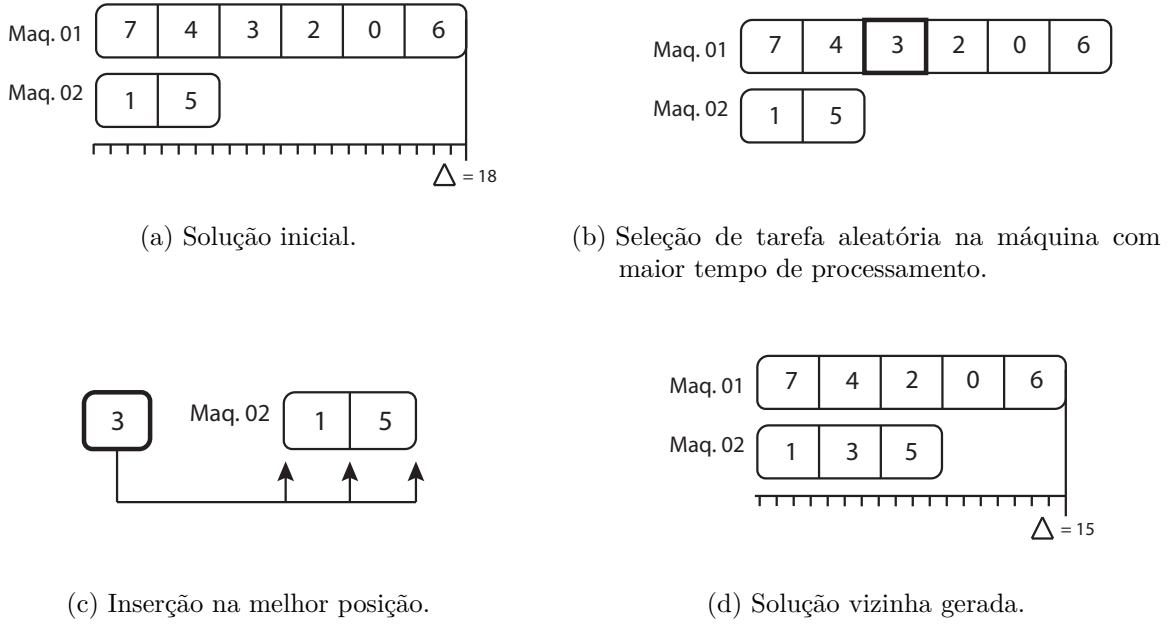


Figura 10 – Busca local por inserção de tarefas.

4.5.2 Busca local por troca de tarefas

Embora o tempo de processamento de uma tarefa seja o mesmo em qualquer uma das máquinas, o número de trocas de ferramentas geradas pela inserção de uma tarefa em uma determinada máquina é dependente de toda a sequência. Assim, a alocação de uma mesma tarefa em uma máquina diferente pode contribuir de forma variada para a função objetivo do IPMTC. Sabendo disso, esta busca local tenta minimizar o *makespan* trocando as tarefas alocadas à máquina com o maior tempo de processamento por tarefas alocadas às demais máquinas. A minimização pode ocorrer pela troca de uma tarefa por outra de menor tempo de processamento ou pela diminuição do número total de trocas de ferramentas.

Para este fim, as máquinas que compõem uma solução são ordenadas de forma não crescente pelo tempo de processamento total. Seja T_i o subconjunto de tarefas alocadas à máquina com o maior tempo de processamento total e T_j o subconjunto de tarefas alocadas à máquina com o menor tempo de processamento total. Para cada uma das tarefas $t_k \in T_i$, o algoritmo seleciona cada uma das tarefas $t_l \in T_j$ como candidata à troca. A ordem em que as tarefas são selecionadas é determinada aleatoriamente. No intuito de diminuir o tamanho da vizinhança induzida por esta busca local, [Fathi e Barnette \(2002\)](#) sugerem verificar se há uma compatibilidade mínima entre as ferramentas requeridas pela tarefa selecionada para inserção e a máquina de destino. Esta verificação desconsidera as ferramentas utilizadas exclusivamente pelas tarefas selecionadas e exige um mínimo de 50% de compatibilidade entre tarefa e máquina de destino.

Havendo compatibilidade em ambas as máquinas, t_k e t_l são trocadas e a nova solução avaliada. Caso o *makespan* seja inferior ao da solução corrente, a solução é atualizada e o processo se reinicia. Caso contrário, a troca é desfeita e move-se para a próxima tarefa t_l . Caso não haja nenhuma troca entre t_k e outra tarefa o algoritmo seleciona a próxima tarefa t_k e o processo repete-se. O algoritmo se encerra quando nenhuma troca entre tarefas produz solução melhor que a atual.

O Algoritmo 2 ilustra as etapas desta busca local. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como a solução corrente s' . As máquinas com o maior e o menor tempo de processamento são identificadas (linhas 2 e 3). Para cada tarefa alocada à máquina com o maior tempo de processamento, a busca local tenta encontrar uma tarefa, entre as alocadas à máquina com o menor tempo de processamento, cuja troca produza um menor valor de *makespan* (linhas 4 a 16). Visando poupar recursos computacionais, antes de efetuar a troca entre duas tarefas selecionadas, a busca local verifica se as ferramentas utilizadas por estas tarefas possuem um mínimo de 50% de similaridade com as ferramentas necessárias ao processamento das demais tarefas alocadas à máquina de destino (linha 6). Havendo compatibilidade as tarefas são trocadas (linha 7) e o tempo de processamento das máquinas envolvidas é comparado ao *makespan* (linha 8). Se ambos forem menor, a solução é atualizada (linha 9) e o método prossegue para a próxima iteração (linha 10). Não havendo compatibilidade, o movimento é descartado (linha 12). A busca local encerra quando não forem encontradas trocas que melhorem o *makespan*.

A Figura 11 ilustra um movimento de troca hipotético desta busca local. Em (a) tem-se uma matriz binária representando as ferramentas necessárias para o processamento de um conjunto tarefas. Em (b) é apresentada uma solução viável composta por duas máquinas sendo a máquina 02 a que possui maior tempo de processamento. Em (c) é demonstrada a seleção aleatória de duas tarefas como candidatas a troca. Na máquina 02 é selecionada a tarefa 3 e na máquina 01, a tarefa 1. Depois de removida a tarefa 1, apenas a ferramenta 1 está instalada na máquina 01. Entretanto, a tarefa 3 requer as ferramentas 2, 3, nenhuma delas instaladas na máquina de destino. As tarefas selecionadas não possuem a compatibilidade de ferramentas exigidas, logo, o movimento é descartado. Na sequência, a busca local seleciona uma nova tarefa na máquina 01. Como visto em (d) a tarefa 4 é selecionada. Ambas as tarefas possuem a compatibilidade de ferramentas exigida e a troca pode ser efetuada. Em (e) é apresentada a solução vizinha gerada pelo movimento e sua avaliação.

Algoritmo 2: Busca local por troca de tarefas.

Entrada: Solução s

Saída: Solução s

```

1  início
2  |    $m_c \leftarrow \arg \max_{m \in [1, \dots, |M|]} \Delta_m$ ;
3  |    $m_f \leftarrow \arg \min_{m \in [1, \dots, |M|]} \Delta_m$ ;
4  |   para cada tarefa  $t_i \in m_c$  em ordem aleatória faça
5  |       para cada tarefa  $t_j \in m_f$  em ordem aleatória faça
6  |           se critério de compatibilidade de ferramentas é atendido então
7  |               troque  $t_i$  e  $t_j$ ;
8  |               se  $\Delta_{m_c} < \Delta$  e  $\Delta_{m_f} < \Delta$  então
9  |                   atualiza  $s$ ;
10 |                   volta à linha 4;
11 |               senão
12 |                   desfaz o movimento;
13 |               fim
14 |           fim
15 |       fim
16 |   fim
17 |   retorna  $s$ ;
18 fim

```

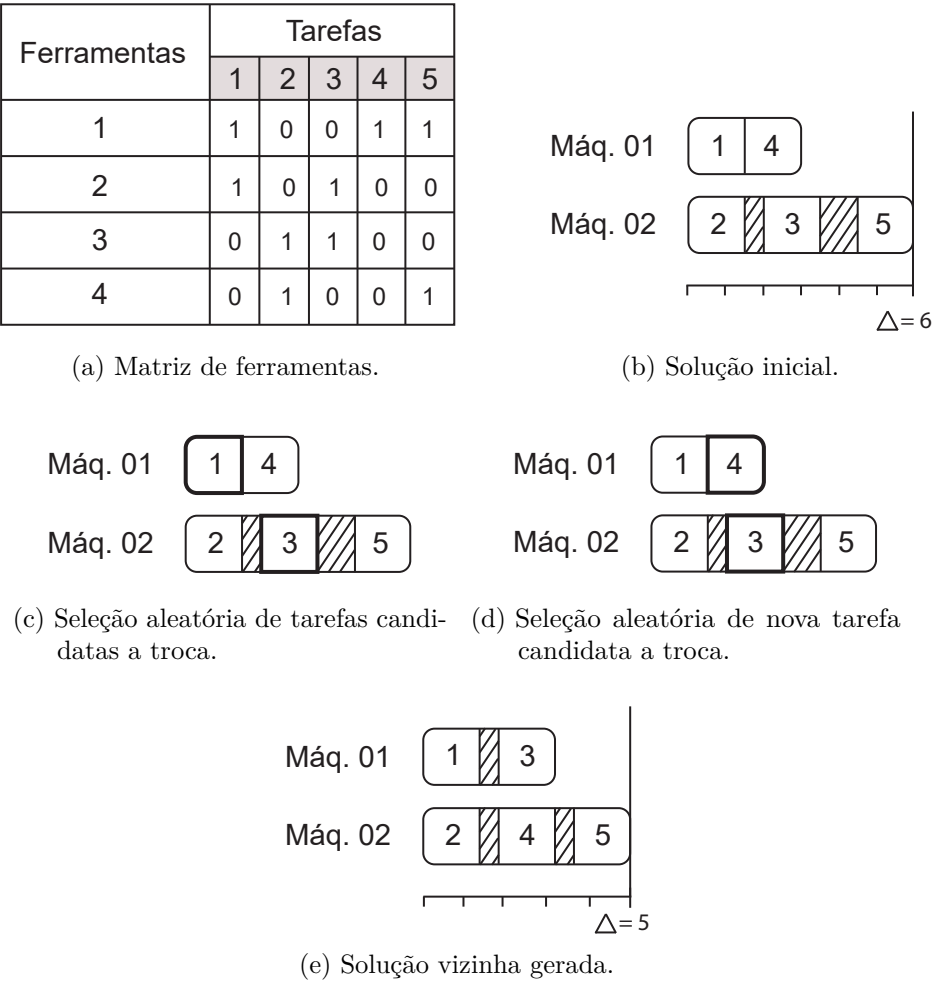


Figura 11 – Busca local por troca de tarefas.

4.5.3 Busca local por agrupamento de blocos de uns

Baseado no conceito de blocos de uns consecutivos, [Paiva e Carvalho \(2017\)](#) apresentam uma busca local que visa diminuir o número de trocas de ferramentas pelo agrupamento destes blocos. No IPMTC, um bloco de uns consecutivos indica o intervalo em que uma ferramenta permaneceu no *magazine* da máquina, tal que dois blocos em uma mesma linha da matriz podem indicar uma troca de ferramentas. A não retirada de ferramentas que serão necessárias em tarefas posteriores reduz o número de trocas de ferramentas, assim, torna-se interessante minimizar o número de blocos de uns.

Para a implementação desta busca local, as máquinas são ordenadas de forma não crescente pelo tempo de processamento. A matriz de ferramentas da máquina com o maior tempo de processamento é analisada linha a linha, em ordem aleatória, a procura de dois ou mais blocos de uns. Para cada par destes blocos, o algoritmo move as colunas do primeiro bloco, uma a uma, para a esquerda e para a direita do segundo bloco, no intuito de agrupá-los. Cada movimento é analisado utilizando-se a avaliação δ apresentada por [Haddadi et al. \(2015\)](#), que retorna a quantidade de blocos de uns criados pelo movimento realizado.

Havendo aumento na quantidade total de blocos de uns, o movimento é claramente prejudicial à solução e é descartado. Entretanto, a diminuição do número total de blocos de uns, devido a utilização da política KTNS, não representa obrigatoriamente uma diminuição no número de trocas de ferramentas. Sendo assim, os movimentos que não aumentam o número de blocos de uns são reavaliados, via KTNS. Dentre os dois possíveis movimentos (à esquerda ou à direita do segundo bloco de uns), é selecionado aquele que gerar maior redução no número de trocas de ferramentas. Em caso de empate, opta-se pela manutenção do último movimento avaliado. Se nenhum movimento ocasionar melhoria na solução, nenhum movimento é realizado. A cada melhoria, reinicia-se a busca local. Caso não haja mais melhoria, a busca local encerra-se.

O Algoritmo 3 ilustra as etapas desta busca local. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como solução corrente. O laço compreendido entre as linhas 2 e 27 coordena a aplicação da busca local enquanto houver melhoria. Inicialmente, não há melhoria (linha 3) e a máquina com maior de processamento é identificada (linha 4). A matriz de ferramentas desta máquina (linha 5) é analisada linha a linha em ordem aleatória (linha 6). Existindo blocos de uns na linha analisada, 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 14). Cada permutação é avaliada primeiramente pela avaliação δ (linha 11 e 15). Se a permutação resultar em aumento do número de blocos de uns ela é imediatamente descartada. Caso contrário, a permutação é avaliada utilizando-se a política KTNS (linha 12 e 16). Ao final de cada avaliação, a coluna é mantida na posição em que houver o menor número de trocas de ferramentas, considerando-se a posição original e as duas permutações

(linha 18). Após a análise total da matriz de ferramentas, havendo o algoritmo encontrado uma solução melhor que a inicial (linha 23), a solução atual é atualizada, as máquinas reordenadas e o processo se reinicia (linhas 23 a 26). O algoritmo encerra quando não forem encontradas permutações que melhorem a solução corrente.

Algoritmo 3: Busca local por agrupamento de blocos de uns.

Entrada: Solução s
Saída: Solução s

```

1 início
2   repita
3     melhoria  $\leftarrow$  falso;
4      $m_c = \arg \max_{m \in [1, \dots, |M|]} \Delta_m$ ;
5      $Q \leftarrow$  matriz de ferramentas de  $m_c$ ;
6     para cada linha  $r \in Q$  selecionada aleatoriamente faça
7       Localize o primeiro 1-bloco  $i \in r$ ;
8       enquanto existir um próximo 1-bloco  $j \in r$  faça
9         para cada coluna  $k \in i$  faça
10           $e \leftarrow$  mova  $k$  para esquerda de  $j$ ;
11          se  $\delta(e) \leq 0$  então
12            | Avalie o número de trocas de ferramentas com o KTNS;
13          fim
14           $d \leftarrow$  mova  $k$  para direita de  $j$ ;
15          se  $\delta(d) \leq 0$  então
16            | Avalie o número de trocas de ferramentas com o KTNS;
17          fim
18          posicione  $K$  onde houver o menor número de trocas de ferramentas
              considerando a posição inicial e as duas permutações;
19        fim
20         $i \leftarrow j$ ;
21      fim
22    fim
23    se  $\Delta_{m_c} < \Delta$  então
24      | melhoria  $\leftarrow$  verdadeiro;
25      | atualize  $s$ ;
26    fim
27  até melhoria = falso;
28  retorna  $s$ ;
29 fim

```

A Figura 12 ilustra um movimento de agrupamento de blocos de uns hipotético desta busca local. Em (a) tem-se a matriz de ferramentas da máquina com o maior tempo de processamento em uma dada solução viável e o número de trocas de ferramentas calculado pela política KTNS. Em (b) é demonstrada a seleção aleatória de uma linha para análise. A linha selecionada possui dois blocos de uns e 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), a avaliação δ do movimento à esquerda confirma a diminuição do

número de blocos de uns. O movimento é avaliado pela política KTNS que demonstra não haver alteração no número de trocas de ferramentas. O movimento à direita, conforme demonstrado em (d), quando avaliado por δ resulta no aumento do número de blocos de uns e por isso é descartado. Como nenhum movimento resultou em diminuição do número de trocas de ferramentas, nenhum movimento é realizado e a matriz é mantida em seu estado original, conforme visto em (a).

Ferramentas	Tarefas			
	3	1	6	4
1	0	1	0	0
2	1	0	1	0
3	0	0	1	1
4	0	0	1	1

KTNS = 1

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

Ferramentas	Tarefas			
	3	1	6	4
1	0	1	0	0
2	1	0	1	0
3	0	0	1	1
4	0	0	1	1

KTNS = 1

(b) Seleção aleatória de linha.

Ferramentas	Tarefas			
	1	3	6	4
1	1	0	0	0
2	0	1	1	0
3	0	0	1	1
4	0	0	1	1

KTNS = 1
 $\delta = -1$

(c) Movimento à esquerda.

Ferramentas	Tarefas			
	1	6	3	4
1	1	0	0	0
2	0	1	1	0
3	0	1	0	1
4	0	1	0	1

KTNS = ?
 $\delta = +1$

(d) Movimento à direita.

Figura 12 – Busca local por agrupamento de blocos de uns.

4.5.4 Descida em vizinhança variável

O método Descida em Vizinhança Variável (ou *Variable Neighborhood Descent*, VND), proposto por [Mladenović e Hansen \(1997\)](#), é uma metaheurística que consiste na mudança sistemática de estrutura de vizinhança durante a exploração do espaço de soluções. A partir de uma dada solução, as vizinhanças são exploradas de forma sequencial. Sempre que uma solução vizinha possuir melhor valor de avaliação, o método reinicia a exploração a partir da primeira vizinhança. Não encontrando solução vizinha com melhor valor de avaliação, o método prossegue para próxima vizinhança. O método encerra quando

todas as vizinhanças forem exploradas sem que uma solução melhor seja encontrada. Desta maneira, uma solução ótima local comum a todas as vizinhanças é retornada.

O Algoritmo 4 demonstra as etapas deste método conforme implementado para organizar a buscas local descritas anteriormente para abordagem do IPMTC. Uma solução viável s é fornecida como parâmetro de entrada e utilizada como solução corrente. O laço compreendido entre as linhas 3 e 19 coordena a exploração das vizinhanças enquanto soluções com melhores valores forem encontradas. As vizinhanças são exploradas sequencialmente. A primeira vizinhança explorada é a Busca Local por Inserção de Tarefas (linhas 4 a 6). Em seguida são exploradas as vizinhanças de Busca Local por Troca de Tarefas (linhas 7 a 9) e Busca Local por Agrupamento de Blocos de Uns (linhas 10 a 12). O *makespan* da solução vizinha s' obtida pela exploração destas vizinhanças é comparada com o *makespan* de s (linha 13). Caso seja menor, o valor de s é atualizado e volta-se a explorar a primeira vizinhança (linhas 14 e 15). Caso contrário, move-se para a próxima vizinhança (linha 17).

Algoritmo 4: Descida em vizinhança variável.

```

Entrada: Solução  $s$ 
Saída: Solução  $s$ 
1  início
2       $k = 1$ ;
3      repita
4          se  $k = 1$  então
5               $s' \leftarrow \text{Execute busca local por inserção de tarefas em } s$ ;
6          fim
7          se  $k = 2$  então
8               $s' \leftarrow \text{Execute busca local por troca de tarefas em } s$ ;
9          fim
10         se  $k = 3$  então
11              $s' \leftarrow \text{Execute busca local por agrupamento de blocos de uns em } s$ ;
12         fim
13         se makespan de  $s' <$  makespan de  $s$  então
14              $s = s'$ ;
15              $k = 1$ ;
16         senão
17              $k = k + 1$ ;
18         fim
19     até  $k = 4$ ;
20     retorna  $s$ ;
21 fim

```

A Figura 13 ilustra parcialmente uma aplicação hipotética do VND sobre uma dada solução viável. Em (a) tem-se uma dada solução viável com *makespan* igual a nove unidades de tempo. Conforme demonstrado em (b), o VND explora a primeira vizinhança, Busca Local por Inserção de Tarefas, e encontra uma solução vizinha com menor valor de

makespan. A tarefa 4 é movida da máquina 01 para a máquina 02. Como houve melhoria na solução, a primeira vizinhança é explorada novamente. Entretanto, nesta nova etapa exploratória nenhuma solução vizinha com menor valor de *makespan* é encontrada. O VND prossegue para a próxima vizinhança, Busca Local por Troca de Tarefas. Conforme demonstrado em (c), uma solução vizinha com menor valor de *makespan* é encontrada. A tarefa 4 passa a ser processada pela máquina 01 e a tarefa 1 passa a ser processada pela máquina 02. Como resultado, o VND volta a explorar a primeira vizinhança mas não encontra solução vizinha com menor valor de *makespan*. O VND prossegue para a segunda vizinhança mas também não encontra solução vizinha com menor valor de *makespan*. A terceira vizinhança, Busca Local por Agrupamento de Blocos de Uns é explorada e, conforme demonstrado em (d), encontra-se uma nova solução vizinha com menor valor de *makespan*. As posições das tarefas 4 e 5 são alteradas na máquina 01. Como foi encontrada uma nova solução com menor valor de *makespan*, o VND é reiniciado.

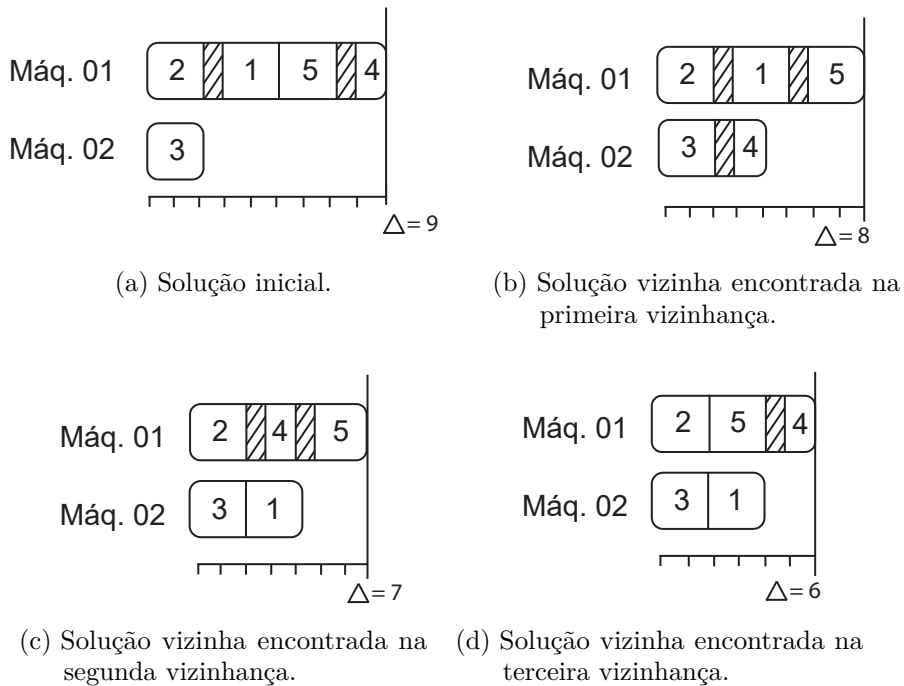


Figura 13 – Descida em vizinhança variável.

A ordem de aplicação das buscas locais foi definida buscando-se o melhor aproveitamento das características intrínsecas de cada uma. Primeiramente, visando um melhor equilíbrio de carga entre as m máquinas, é aplicado a busca local por inserção de tarefas. Essa busca local possui a tendência de reduzir o *makespan* com maior ênfase, resultando em soluções equilibradas nas quais não há margem para novas inserções simples. Após essa etapa, a busca local por troca de tarefas é aplicada, cuja característica intrínseca é primeiro remover uma tarefa para depois inserir outra, ou seja, cria-se margem para uma inserção cruzada entre duas máquinas diferentes. Por fim, a margem para movimentação de tarefas entre máquinas é ainda mais reduzida. Portanto, optou-se por concentrar a

busca na característica de trocas de ferramentas do IPMTC, por vezes negligenciado. Especificamente, busca-se minimizar apenas as trocas de ferramentas em uma mesma máquina empregando-se a busca local por agrupamento de blocos de uns.

Conforme descrito no Capítulo 3, os cromossomos pertencentes ao conjunto elite são copiados integralmente de uma geração para a outra. Soma-se a isto o fato de que as buscas locais resultam em um ótimo local que não se altera caso aplique-se o mesmo procedimento novamente. Assim, para evitar a execução desnecessária das buscas locais, uma vez aplicadas sem melhoria, o cromossomo é marcado e ignorado em caso de reincidência.

A aplicação do VND indistintamente a todos os cromossomos pode corroborar para uma convergência prematura e aumentar significativamente o tempo de execução do BRKGA. Assim, no Capítulo 5, foram realizados experimentos preliminares que possibilitaram determinar a melhor forma de aplicação das buscas locais fazendo-se um *tradeoff* entre qualidade de solução e tempo de execução.

5 Experimentos computacionais

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

5.1 Conjuntos de instâncias

As 2880 instâncias disponibilizadas por [Beezão et al. \(2017\)](#) são utilizados nos experimentos. As instâncias estão separadas em dois conjuntos contendo 1440 instâncias cada. O conjunto IPMTC-I foi gerado com base no método de geração de [Yanasse \(2009\)](#) para instâncias do MTSP. Para instâncias com 8 tarefas, o número de máquinas é 2. Para instâncias com 15 tarefas, o número de máquinas varia de 2 a 3. Para instâncias com 25 tarefas, o número de máquinas varia de 2 a 4. O número de ferramentas para esse conjunto de instâncias varia entre 15 e 20.

O conjunto IPMTC-II contém instâncias maiores. Para instâncias com 50 tarefas, o número de máquinas varia entre 3 e 5. Para instâncias com 100 tarefas, o número de máquinas varia entre 4 e 7. Para instâncias com 200 tarefas, o número de máquinas varia entre 6 e 10. O número de ferramentas para esse conjunto de instâncias varia entre 30 e 40.

Devido ao grande número de instâncias em ambos os conjuntos, seguindo o mesmo padrão utilizado por [Beezão et al. \(2017\)](#), os resultados apresentados são agrupados pela quantidade de máquinas, tarefas e ferramentas. Assim, cada linha nas tabelas que apresentam resultados dos experimentos contém os valores médios das soluções individuais das instâncias pertencentes àquele subgrupo. Para o conjunto IPMTC-I, cada subgrupo corresponde a 120 instâncias e para o conjunto IPMTC-II, cada subgrupo corresponde a 60 instâncias.

5.2 Experimentos preliminares

Para viabilizar a comparação do método proposto com o estado da arte atual, alguns experimentos preliminares foram realizados. Estes experimentos compreendem a implementação e avaliação de uma formulação de programação linear inteira e também da metaheurística ALNS. A seguir, ambos os experimentos são reportados em detalhes.

5.2.1 Formulação matemática para o IPMTC

O subconjunto de instâncias utilizados nos experimentos das formulações matemáticas reportados originalmente por Beezão et al. (2017) não é detalhado pelos autores. Os valores das soluções ótimas obtidas também não são fornecidos. Assim, os experimentos realizados com estas formulações pouco contribuem para a continuidade dos estudos do IPMTC.

No intuito de obter as soluções ótimas para as instâncias consideradas neste trabalho e obter um balizamento quanto à qualidade das soluções, o modelo M_1 de Beezão et al. (2017), detalhado na Seção 3.1.1, foi implementado. Utilizou-se a linguagem *python* versão 2.7.12 e o *solver Gurobi* versão 8.0.1. O ambiente computacional adotado para os experimentos consiste em um computador com processador *Intel i5 Quad Core* de 2.3 GHz com 8 GB de memória RAM sob o sistema operacional Ubuntu 17.10. O tempo máximo de execução do *solver* foi configurado em 4200 segundos, de acordo com o experimento original.

A Tabela 5 detalha os resultados obtidos para cada subconjunto de instâncias do conjunto IPMTC-I. São apresentados o número de máquinas (m), o número de tarefas (n), o número de ferramentas (l), o valor médio das soluções (S), o percentual de soluções ótimas encontradas ($OPT(\%)$), o *gap* em relação à relaxação linear ($gap(\%)$) e o tempo médio de execução (T), em segundos.

Tabela 5 – Resultados obtidos pelo modelo M_1 para o IPMTC-I.

m	n	l	S	$OPT(\%)$	$gap(\%)$	T
2	8	15	227,49	100,00	0,00	14,42
2	8	20	250,70	100,00	0,00	36,98
2	15	15	390,78	0,00	31,23	4200,86
2	15	20	384,07	25,00	28,28	3151,69
2	25	15	460,38	0,00	15,63	4200,07
2	25	20	553,81	0,00	25,54	4200,02
3	15	15	251,48	10,00	26,72	3784,25
3	15	20	230,49	25,00	22,04	3150,62
3	25	15	227,12	0,00	6,71	4200,03
3	25	20	333,32	0,83	20,16	4166,10
4	25	15	191,20	24,17	0,70	3202,88
4	25	20	218,41	6,67	13,29	3925,44

Assim como reportado originalmente, para instâncias com 8 tarefas o modelo alcançou todos os resultados ótimos. Entretanto, para instâncias maiores, o modelo encontrou dificuldade já a partir das instâncias com 15 tarefas. No total, para o conjunto IPMTC-I, o modelo encontrou resultados ótimos para 24,31% das instâncias. Para as instâncias do conjunto IPMTC-II, o modelo sequer resolve a relaxação linear dentro do

limite de tempo estipulado. Os resultados obtidos para o conjunto IPMTC-I obtidos neste experimento serão considerados nas comparações da Seção 5.3.

5.2.2 Configuração e validação da implementação do ALNS

Em seu trabalho original, Beezão et al. (2017) compara diferentes versões do método ALNS às heurísticas CLIP1 e CLIP2 (FATHI; BARNETTE, 2002), que então representavam o estado da arte. Entretanto, não foram fornecidos os valores para as soluções obtidas. Apenas a distância percentual em relação à melhor solução comparada é informada, indicando que a versão ALNS-R superou as duas heurísticas anteriores. Desta forma, não seria possível comparar os resultados obtidos neste trabalho com o estado da arte atual. Mediante solicitação, parte dos resultados do método ALNS-R, referentes ao conjunto IPMTC-II, foi disponibilizado. Para cada instância do conjunto IPMTC-II, o método ALNS-R foi executado dez vezes, entretanto, para algumas instâncias, os dados não correspondem a esta quantidade e 43 instâncias não possuem resultados. No total, 3,51% dos resultados estão ausentes.

Para fornecer uma comparação adequada em termos de arquitetura de *hardware* e possibilitar a comparação com o conjunto de instâncias IPMTC-I, o método ALNS, conforme descrito pelos autores, foi implementado. Os resultados originais fornecidos foram utilizados como balizamento para validar a implementação realizada. O ambiente computacional adotado para os experimentos computacionais consiste em 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. Assim como na versão de Beezão et al. (2017), o método ALNS foi implementado utilizando-se a linguagem C++.

Em seus experimentos, Beezão et al. (2017) utilizam seis cenários de configurações do ALNS, variando o número de iterações e a quantidade de tarefas removidas pelas heurísticas de remoção. Entre os cenários apresentados, os autores relatam que o número quatro apresenta os melhores resultados. Neste cenário, o limite de iterações do método é igual a 25000 e o percentual de tarefas removidas é limitado em 25%. Os resultados parciais originais fornecidos pertencem a este cenário, os quais demonstram que o critério de parada mais frequente é o limite do tempo de execução de 14400 segundos, ou 4 horas. Adicionalmente, após a nova implementação do método e análises preliminares, os resultados obtidos mostraram-se de qualidade inferior aos fornecidos pelos autores. Desta forma, alterou-se a configuração para o cenário três, em que o limite de iterações permanece em 25000 e o percentual de tarefas removidas é limitado em 10%. Neste cenário, a nova implementação obteve resultados com qualidade média final superior aos fornecidos pelos autores.

A Tabela 6 apresenta os resultados médios de dez execuções, obtidos pela nova implementação do ALNS e os valores médios fornecidos por Beezão et al. (2017). São

apresentados o número de máquinas (m), o número de tarefas (n), o número de ferramentas (l), a melhor solução encontrada (S^*), a média das soluções encontradas considerando as 10 execuções (S), o desvio padrão obtidos nas 10 execuções para cada instância (σ), o tempo médio de execução (T), em segundos e a distância percentual (gap) entre a melhor solução obtida pela nova implementação proposta e a original fornecida pelos autores, calculada como $gap = \frac{novo-original}{original} \times 100$. Como mencionado anteriormente, alguns resultados de instâncias fornecidos por Beezão et al. (2017) estão ausentes ou contemplam menos execuções do que as 10 originais. Os subconjuntos aos quais tais instâncias pertencem são indicados com *. Os resultados ausentes foram preenchidos com os resultados obtidos pela nova implementação visando completar os dados sem interferir na comparação. Os melhores resultados são apresentados em negrito.

Para fornecer uma comparação justa de tempo de execução, considerando-se as diferentes arquiteturas utilizadas nos experimentos, o tempo obtido pela nova implementação do ALNS, relatado na Tabela 6, foi normalizado segundo o *benchmark* fornecido por Passmark (2018). De acordo com este *benchmark*, a pontuação por *thread* individual do computador utilizado em nossos experimentos é de 1,499 enquanto a do utilizado nos experimentos de Beezão et al. (2017) é de 1,249.

Tabela 6 – Comparação entre as implementações do ALNS.

m	n	l	ALNS - Nova Implementação					ALNS (BEEZÃO et al., 2017)			
			S^*	S	σ	T	$gap(\%)$	S^*	S	σ	T
3	50	30	1372,23	1426,24	16,97	182,44	2,11	* 1343,92	*1402,77	*13,97	*11475,09
3	50	40	1525,28	1577,17	15,57	239,75	1,24	* 1506,57	*1542,46	*12,31	*14123,51
4	50	30	1026,05	1057,87	29,86	164,83	1,70	1008,85	1030,62	11,47	11705,45
4	50	40	1056,62	1079,67	12,72	184,18	2,11	1034,80	1059,30	12,29	14311,55
4	100	30	2191,35	2251,69	23,65	1503,59	-5,50	*2318,77	*2401,31	*31,85	*14441,63
4	100	40	2712,32	2754,51	25,45	2325,65	-7,02	*2917,20	*2987,22	*42,87	*14482,89
5	50	30	830,73	905,97	13,15	120,07	1,28	* 820,27	*861,15	*8,75	*10854,76
5	50	40	803,43	825,06	12,06	156,23	3,62	775,33	792,93	9,88	14044,71
5	100	30	1763,27	1801,05	21,19	1254,79	-5,96	1874,95	1921,41	25,91	14446,19
5	100	40	2254,35	2301,59	25,24	1962,76	-7,42	2435,08	2494,69	35,54	14482,06
6	100	30	1526,93	1568,21	21,44	1067,27	-4,16	*1593,25	*1658,19	*23,87	*14439,15
6	100	40	1951,77	2000,67	26,14	1638,56	-6,87	2095,85	2158,14	35,37	14485,32
6	200	30	3014,58	3090,23	56,64	12912,04	-6,17	3212,97	3291,48	48,10	14524,61
6	200	40	3634,65	3709,35	95,01	17131,60	-5,78	3857,63	3988,87	77,45	14633,73
7	100	30	1269,10	1303,32	19,24	941,33	-3,58	1316,23	1346,65	17,29	14428,71
7	100	40	1779,07	1825,82	26,22	1483,09	-2,54	1825,48	1869,58	23,42	14412,62
7	200	30	2506,73	2572,84	36,96	11186,59	-4,56	2626,37	2709,82	49,68	14541,80
7	200	40	3286,83	3351,70	120,01	16238,72	-5,14	3465,02	3586,36	68,73	14622,21
8	200	30	2348,10	2415,46	35,78	10019,96	-4,08	2448,02	2517,24	42,99	14534,02
8	200	40	2882,42	2938,67	72,01	14473,07	-4,66	3023,32	3118,65	59,87	14624,42
9	200	30	1985,87	2045,51	34,17	8900,35	-2,68	2040,53	2102,57	37,30	14531,10
9	200	40	2484,77	2546,25	35,32	13089,10	-2,85	2557,75	2647,69	55,06	14637,44
10	200	30	1954,43	2012,95	33,25	8197,56	-2,07	1995,75	2049,91	33,43	14528,04
10	200	40	2208,63	2271,82	36,29	12011,48	-2,30	2260,57	2332,37	41,69	14637,08

De acordo com os dados reportados, a nova implementação do ALNS, com a alteração de cenário, obteve um gap médio de -3,19% quando comparado aos resultados

originais. O tempo de execução da nova implementação foi 59,35% menor, considerando os tempos de execução já convertidos. Nas comparações descritas na Seção 5.3 são utilizados os resultados obtidos pela nova implementação do ALNS, exceto para os seis subconjuntos de instâncias em que os resultados originais fornecidos são melhores:

- $m = 3, n = 50, l = 30$;
- $m = 3, n = 50, l = 40$;
- $m = 4, n = 50, l = 30$;
- $m = 4, n = 50, l = 40$;
- $m = 5, n = 50, l = 40$; e
- $m = 5, n = 50, l = 40$.

A manutenção de parte dos resultados fornecidos é realizada no intuito de garantir que os melhores resultados conhecidos do ALNS sejam utilizados nas comparações reportadas nas seções seguintes.

5.2.3 Ajuste de parâmetros do BRKGA

Os parâmetros do BRKGA foram definidos utilizando-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 uma combinação apropriada de valores para os parâmetros. A Tabela 7 demonstra os valores considerados pelo *irace* na definição dos parâmetros e os valores selecionados por este experimento.

Tabela 7 – Valores considerados e selecionados pelo *irace* para definição dos parâmetros.

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

Buscando um melhor aproveitamento da arquitetura computacional atualmente disponível, os processos de decodificação e aplicação das buscas locais foram paralelizados. Para evitar uma disparidade excessiva na comparação dos métodos, visto que o ALNS é executado de forma sequencial, limitou-se o número máximo de *cores* utilizados pelo BRKGA a quatro.

Adicionalmente, visando alcançar soluções de alta qualidade em baixo tempo computacional, foram definidos, empiricamente, as formas de aplicação das buscas locais

e os valores para os critérios de parada. Visando evitar a convergência prematura para ótimos locais e um alto consumo de tempo, definiu-se uma probabilidade de 5% para a aplicação das buscas locais. Assim, para cada cromossomo é sorteado um número aleatório no intervalo $[0,1]$ e as buscas locais só são aplicadas nos casos em que esse número é maior que 0,95. Somado a isso, foram adotadas estratégias diferentes para a aplicação das buscas locais nos conjuntos de cromossomos elite e não-elite. Para os cromossomos pertencentes ao conjunto elite, optou-se por utilizar o método de Descida em Vizinhança Variável, descrito anteriormente na Seção 4.5.4. Para os cromossomos pertencentes ao conjunto não-elite, buscando novamente evitar a convergência prematura e possíveis ótimos locais, optou-se por aplicar apenas uma vez cada uma das buscas locais, por geração. O número máximo de gerações foi definido como 100 e o tempo máximo de execução limitado a 3600 segundos.

5.3 Comparação com o estado da arte

Após a realização dos experimentos preliminares, definiu-se a versão final do BRKGA. No experimento relatado nesta seção, comparam-se os resultados obtidos por este método aos resultados obtidos pelo ALNS (BEEZÃO et al., 2017). O ambiente computacional adotado para os experimentos computacionais é o mesmo descrito na Seção 5.2.2. 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 IPMTC, compilado com g++ 7.2.0 e a opção de otimização -O3.

A Tabela 8 apresenta os resultados médios de dez execuções dos métodos para cada instância do conjunto IPMTC-I. São apresentados o número de máquinas (m), o número de tarefas (n) e o número de ferramentas (l). Para os métodos BRKGA e ALNS são apresentados o valor das melhores soluções (S^*), o valor médio das soluções (S), o desvio padrão (σ) e o tempo médio de execução (T), em segundos. Para o modelo M_1 é apresentado o valor médio das soluções (S^*). É apresentada ainda a distância percentual (*gap*) entre a melhor solução do BRKGA e a melhor solução conhecida, seja do ALNS ou do modelo M_1 , calculada como $gap = \frac{BRKGA - \min\{ALNS, M_1\}}{\min\{ALNS, M_1\}} \times 100$. Os melhores resultados são destacados em negrito.

Conforme mencionado na Seção 5.2.1, o modelo M_1 reportou resultados ótimos para 24,31% das instâncias do conjunto IPMTC-I. O BRKGA encontrou resultados equivalentes a todos os ótimos reportados, enquanto o ALNS encontrou soluções equivalentes às ótimas para 91,43% deste ótimos. Além disso, o ALNS não foi capaz de superar os resultados médios alcançados pelo modelo M_1 em um subconjunto de instâncias ($m = 4, n = 25, l = 15$). O método proposto gerou soluções médias de qualidade igual ou superior para todos os subconjuntos de instâncias. Para o subconjunto de instâncias em que registrou-se a maior diferença entre as soluções ($m = 4, n = 25, l = 20$), o *gap* médio foi de -0,71% e as

Tabela 8 – Resultados para o grupo IPMTC-I.

m	n	l	BRKGA					ALNS				M_1
			S^*	S	σ	T	$gap(\%)$	S^*	S	σ	T	S^*
2	8	15	227,49	227,50	0,02	0,03	0,00	227,49	228,53	1,57	46,38	227,49
2	8	20	250,70	250,83	0,30	0,02	0,00	250,70	251,66	1,53	48,49	250,70
2	15	15	367,22	368,04	0,95	0,48	0,00	367,22	368,81	4,40	3,32	390,77
2	15	20	370,08	370,65	0,63	0,55	0,00	370,08	370,36	0,39	3,90	384,06
2	25	15	454,10	454,14	0,06	2,87	0,00	454,11	454,71	0,59	8,65	460,38
2	25	20	510,24	510,33	0,10	3,92	-0,02	510,30	511,23	0,94	4,78	533,81
3	15	15	230,53	231,24	0,82	0,20	-0,02	230,56	231,06	0,60	2,87	251,48
3	15	20	217,92	218,50	0,55	0,22	0,00	217,92	218,31	0,54	3,28	230,49
3	25	15	270,00	271,30	1,48	1,05	-0,28	270,77	274,74	3,08	6,86	277,13
3	25	20	306,48	307,45	0,88	1,40	-0,38	307,65	312,10	3,57	9,52	333,32
4	25	15	189,99	190,29	0,30	0,56	-0,16	191,49	193,23	1,13	6,07	191,20
4	25	20	200,83	204,12	2,59	0,71	-0,70	202,24	206,44	3,04	8,15	218,41

maiores diferenças ocorrem nas maiores instâncias do conjunto. O gap máximo reportado entre todas as instâncias deste conjunto foi de 6,55% e o mínimo de -6,88%. Novos melhores resultados foram encontrados para 315 (ou 21,88%) instâncias deste conjunto. Se considerarmos resultados melhores ou iguais, o método proposto obteve os melhores resultados para 1429 (ou 99,24%) instâncias deste conjunto.

Em média, o tempo computacional médio do BRKGA foi 92,10% menor do que o tempo computacional médio do método ALNS. O tempo médio de execução do BRKGA para todas as instâncias deste conjunto foi de 1,01 segundos, evidenciando a facilidade do método na abordagem de tais instâncias. O desvio padrão médio de 0,72 para o BRKGA e de 1,78 para o ALNS demonstra consistência dos métodos em gerar soluções com baixa variação em execuções independentes.

Realizou-se adicionalmente uma análise estatística para comparação dos três métodos considerados neste experimento. O teste de normalidade Shapiro-Wilk ([SHAPIRO; WILK, 1965](#)) confirmou, com um intervalo de confiança de 95%, a hipótese nula de que os resultados comparados, BRKGA ($W = 0,88405$ e $p - value = 0,09879$), ALNS ($W = 0,88244$ e $p - value = 0,09418$) e M_1 ($W = 0,89268$ e $p - value = 0,1276$) poderiam ser modelados de acordo com uma distribuição normal. O teste paramétrico de Análise de Variância (ANOVA) foi aplicado para verificar se existe diferença significativa entre os métodos comparados. O teste resultou em $p - value = 0,945$ indicando que não é possível afirmar que há diferença significativa entre os resultados dos métodos para as instâncias do conjunto IPMTC-I.

As buscas locais utilizadas na hibridização do método proposto, em média, melhoraram a qualidade das soluções do conjunto IPMTC-I em 0,43%. Dentre essas, a busca local por inserção de tarefas é a que mais contribui para a qualidade da solução. Em média, para o conjunto IPMTC-I, considerando-se apenas as melhorias geradas pelas buscas locais, a

busca local por inserção de tarefa é responsável por 60,91% das melhorias. A busca local por troca de tarefas por 27,19% e a busca local por agrupamento de blocos de uns por 11,90%.

Segundo os dados reportados por [Beezão et al. \(2017\)](#), o tempo gasto com trocas de ferramentas é responsável, em média, por 33,44% do *makespan* obtido pelo ALNS para as instâncias do conjunto IPMTC-I. A baixa influência das trocas de ferramentas no *makespan* e as pequenas dimensões dos parâmetros m , n e l tendem tornar as instâncias menos difíceis de se resolver, pois minimizam as características intrínsecas ao problema. Esta propriedade justifica o equilíbrio reportado entre os métodos comparados, neste conjunto de instâncias, justificando a necessidade de um segundo conjunto de instâncias.

A Tabela 9 apresenta os resultados obtidos para as instâncias pertencentes ao grupo IPMTC-II. Os dados apresentados seguem o mesmo padrão da Tabela 8. Conforme descrito na Seção 6, os melhores resultados conhecidos para o método ALNS são utilizados. Assim, para as conjuntos de instâncias $m = 3, n = 50, l = 30$; $m = 3, n = 50, l = 40$; $m = 4, n = 50, l = 30$; $m = 4, n = 50, l = 40$; $m = 5, n = 50, l = 40$ e $m = 5, n = 50, l = 40$ são utilizados os resultados originais fornecidos por [Beezão et al. \(2017\)](#) com o tempo devidamente ajustado de acordo com o *benchmark* fornecido pelo [Passmark \(2018\)](#). Para todos os demais conjuntos, os resultados do ALNS correspondem aos resultados médios obtidos pela nova implementação.

De acordo com os dados reportados, é possível verificar que o método proposto obteve novos melhores resultados médios para 91,67% dos subconjuntos de instâncias pertencentes ao IPMTC-II. Apenas em dois destes ($m = 3, n = 50, l = 40$; e $m = 4, n = 50, l = 40$), o método proposto não foi capaz de gerar novos melhores resultados médios. Entretanto, mesmo para estes subconjuntos, o *gap* médio de 0,61% atesta uma qualidade equiparável entre os métodos. No geral, o método proposto obteve um *gap* médio de -2,39%. O *gap* máximo reportado entre todas as instâncias deste conjunto foi de 6,75% e o mínimo de -12,71%. Novos melhores resultados foram encontrados para 1204 (ou 83,61%) instâncias do conjunto. Os menores valores de *gap* foram registrados para as maiores instâncias. Agrupando-se as instâncias por número de tarefas, o *gap* médio para instâncias com 50 tarefas é de apenas -0,31%. Para instâncias com 100 tarefas, o *gap* médio é de -1,26% e, para instâncias com 200 tarefas, o *gap* médio é de -4,55%.

Em média, o tempo computacional do BRKGA foi 93,27% menor do que o tempo computacional do ALNS. Para o subconjunto de instâncias que demandou maior tempo de processamento ($m = 6, n = 200, l = 40$), a média de tempo do ALNS foi de 14230,64 segundos, enquanto o BRKGA obteve um tempo médio de 2524,16 segundos, apenas 17,74% do tempo exigido pelo ALNS. O desvio padrão médio de 14,98 (ou 0,77%) para o BRKGA e de 35,02 (ou 1,74%) para o ALNS demonstram novamente a consistência dos métodos em gerar soluções com baixa variação em execuções independentes.

Tabela 9 – Resultados para o grupo IPMTC-II.

m	n	l	BRKGA					ALNS			
			S^*	S	σ	T	$gap(\%)$	S^*	S	σ	T
3	50	30	1336,73	1365,00	16,03	87,75	-0,53	1343,92	1402,77	13,97	9524,33
3	50	40	1518,57	1536,11	9,97	118,57	0,80	1506,57	1542,46	12,31	11722,51
4	50	30	996,30	1019,90	12,79	40,42	-1,24	1008,85	1030,62	11,47	9715,52
4	50	40	1039,13	1057,58	10,56	53,78	0,42	1034,80	1059,30	12,29	11878,59
4	100	30	2179,80	2208,43	15,99	185,27	-0,53	2191,35	2251,69	23,65	1252,99
4	100	40	2698,33	2733,59	19,10	297,34	-0,52	2712,32	2757,51	25,45	1938,04
5	50	30	809,83	828,44	9,68	21,83	-1,27	820,27	861,15	8,75	9009,45
5	50	40	775,18	791,89	10,18	27,99	-0,02	775,33	792,93	9,88	11657,10
5	100	30	1743,83	1766,46	12,56	89,71	-1,10	1763,27	1801,05	21,19	1045,66
5	100	40	2237,43	2269,39	16,67	154,31	-0,75	2254,35	2301,59	25,24	1635,63
6	100	30	1497,82	1520,69	12,77	49,78	-1,91	1526,93	1568,21	21,44	889,39
6	100	40	1930,17	1958,68	15,87	84,58	-1,11	1951,77	2000,67	26,14	1365,47
6	200	30	2932,23	2961,92	17,04	1780,59	-2,73	3014,58	3090,23	56,64	10760,03
6	200	40	3539,23	3574,86	19,09	2524,16	-2,63	3634,65	3709,35	95,01	14276,33
7	100	30	1237,33	1257,34	10,67	29,67	-2,50	1269,10	1303,32	19,24	784,44
7	100	40	1749,20	1779,50	15,41	52,93	-1,68	1779,07	1825,82	26,22	1235,91
7	200	30	2410,75	2438,57	14,50	1078,50	-3,83	2506,73	2572,84	36,96	9322,16
7	200	40	3178,30	3217,27	20,06	1711,91	-3,30	3286,83	3351,70	120,01	13532,27
8	200	30	2245,20	2273,57	15,61	658,84	-4,38	2348,10	2415,46	35,78	8349,97
8	200	40	2752,63	2788,63	17,21	1011,76	-4,50	2882,42	2938,67	72,01	12060,89
9	200	30	1880,13	1902,56	12,16	410,69	-5,32	1985,87	2045,51	34,17	7416,96
9	200	40	2346,20	2373,00	14,27	678,48	-5,58	2484,77	2546,25	35,32	10907,58
10	200	30	1827,12	1849,48	12,22	297,65	-6,51	1954,43	2012,95	33,25	6831,30
10	200	40	2059,75	2086,52	13,96	466,93	-6,74	2208,63	2271,82	36,29	10009,57

Adicionalmente, foi realizada uma análise estatística para comparação dos resultados obtidos pelos métodos considerados. O teste de normalidade Shapiro-Wilk ([SHAPIRO; WILK, 1965](#)) confirmou com um intervalo de confiança de 95%, a hipótese nula de que os resultados comparados dos métodos BRKGA ($W = 0,97884$ e $p - value = 0,8736$) e ALNS ($W = 0,97874$ e $p - value = 0,8715$) poderiam ser modelados de acordo com uma distribuição normal. Assim, o método paramétrico Student's t-test, introduzido por [Student \(1908\)](#), foi utilizado para comparar os dois conjuntos de soluções. O t-test indicou que existe diferença significativa entre os resultados e que o BRKGA possui melhores valores médios ($t = -5,0927$, $df = 23$, e $p = 1,854e - 05$, para um nível de significância de 0,05).

Segundo ([BEEZÃO et al., 2017](#)), o tempo gasto com trocas de ferramentas é responsável, em média, por 67,66% do *makespan* obtidos pelo ALNS para as instâncias do conjunto IPMTC-II. Conforme descrito no Capítulo 4, o BRKGA possui operadores específicos para esta característica do IPMTC, que é negligenciada pelos operadores do ALNS e corrobora ativamente para a maior qualidade nas soluções encontradas pelo BRKGA.

As buscas locais utilizadas na hibridização do método proposto, em média, melhoraram a qualidade das soluções do conjunto IPMTC-II em 11,05%. Dentre essas, a busca

local por inserção de tarefas novamente é a que mais contribui para a qualidade da solução. Em média, para o conjunto IPMTC-II, considerando-se apenas as melhorias geradas pelas buscas locais, a busca local por inserção de tarefas é responsável por 68,93% das melhorias. A busca local por troca de tarefas por 27,61% e a busca local por agrupamento de blocos de uns por 3,47%.

Considerando-se os resultados e a análise estatística reportados, pode-se concluir que o método proposto é superior ao estado atual da arte, pois produz resultados de qualidade comparável ou superior em um tempo computacional significativamente inferior para todas as instâncias disponíveis na literatura. A superioridade demonstrada pelo método proposto deve-se principalmente à implementação de operadores específicos para cada característica do IPMTC. Conforme mencionado no Capítulo 1, o IPMTC é oriundo da junção do SPM com o MTSP, em que o tempo de preparação depende da sequência está relacionado com trocas de ferramentas, assim, as características de ambos os problemas precisam ser consideradas na resolução do problema. As buscas locais utilizadas na hibridização do método proposto foram desenvolvidas especificamente para abordar estas características. As buscas locais por inserção de tarefas e por troca de tarefas abordam, principalmente, as características do SPM, procurando obter o melhor sequenciamento possível para as n tarefas entre as m máquinas disponíveis. Já a busca local por agrupamento de blocos de uns explora as características do MTSP ao tentar minimizar o número de trocas de ferramentas em uma máquina. Ao não negligenciar nenhuma característica intrínseca ao problema, o método proposto alcança resultados superiores ao ALNS, tornando-se o novo estado da arte para o IPMTC.

5.3.1 Experimentos adicionais

Para ilustrar uma comparação entre os tempos dos métodos ignorando os efeitos da paralelização utilizada pelo BRKGA, uma nova versão, sem paralelismo, foi executada para um conjunto representativo de 5% das instâncias de ambos os conjuntos IPMTC-I e IPMTC-II.

Os tempos médios de 10 execuções independentes são apresentados na Tabela 10, em que as instâncias com até 25 tarefas são referentes ao conjunto IPMTC-I e as demais são referentes ao conjunto IPMTC-II. São apresentados o número de tarefas (n), o tempo médio do BRKGA paralelo ($BRKGA_p$), o tempo médio do BRKGA sequencial ($BRKGA_s$) e o tempo médio do ALNS (ALNS). Todos os tempos são expressos em segundos.

Embora não se possa generalizar para todo o conjunto de instâncias, os resultados apresentados ilustram que, mesmo implementado sequencialmente, o BRKGA é sensivelmente mais rápido que o ALNS. Para as instâncias selecionadas pertencentes ao conjunto IPMTC-I, o BRKGA sequencial obteve um tempo médio 95,45% menor do que o tempo médio obtido pelo ALNS para as mesmas instâncias. Para as instâncias selecionadas

Tabela 10 – Comparação entre o tempo de processamento do BRKGA paralelo, BRKGA sequencial e ALNS.

n	BRKGA _p	BRKGA _s	ALNS
8	0,02	0,04	52,70
15	0,19	0,52	3,65
25	0,83	2,41	8,97
50	43,88	114,18	200,99
100	220,29	525,53	1348,81
200	1597,86	2866,45	11276,44

pertencentes ao conjunto IPMTC-II, o tempo médio obtido pelo BRKGA sequencial foi 72,66% menor do que o tempo médio obtido pelo ALNS. Considerando-se todas as instâncias selecionadas, o BRKGA paralelo, conforme proposto, foi em média 46,91% mais rápido do que sua versão sequencial.

5.4 Análise de convergência do BRKGA

A convergência média do BRKGA em ambos os conjuntos de instâncias é sumarizada pelo gráfico do tipo *boxplot* apresentado na Figura 14. Para a maior parte das instâncias pertencentes ao conjunto IPMTC-I, o BRKGA converge antes de 20 iterações. Para as instâncias pertencentes ao conjunto IPMTC-II, a maior parte das convergências ocorre abaixo de 50 iterações. Tal diferença explica-se pela disparidade entre os dois conjuntos de instâncias. Para o as instâncias do grupo IPMTC-I observa-se uma forma assimétrica positiva com variabilidade intermediária, ponto máximo próximo a 60 iterações e apenas 3 *outliers* acima do limite superior. As instâncias do grupo IPMTC-II apresentam uma forma simétrica com variabilidade intermediária. O ponto máximo encontra-se próximo a 80 iterações e há 71 *outliers* acima do limite superior. O tempo limite de processamento do BRKGA, fixado em 3600 segundos, foi atingido em apenas 207 (ou 0,72% do total) execuções. Todas as execuções nas quais o tempo limite foi atingido se referem a instâncias com 200 tarefas.

Para ilustrar a convergência do BRKGA foram utilizados gráficos do tipo *time-to-target* (AIEX; RESENDE; RIBEIRO, 2007), ou gráficos ttt. 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. Para uma determinada instância, o BRKGA foi executado 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 para a instância. As instâncias foram agrupadas por número de tarefas e, aleatoriamente, selecionou-se uma tarefa de cada grupo para geração dos gráficos ttt. Em seguida, comparou-se a distribuição resultante (*empirical*, representada pelas cruces roxas) com a distribuição exponencial (*theoretical*, representada pela linha

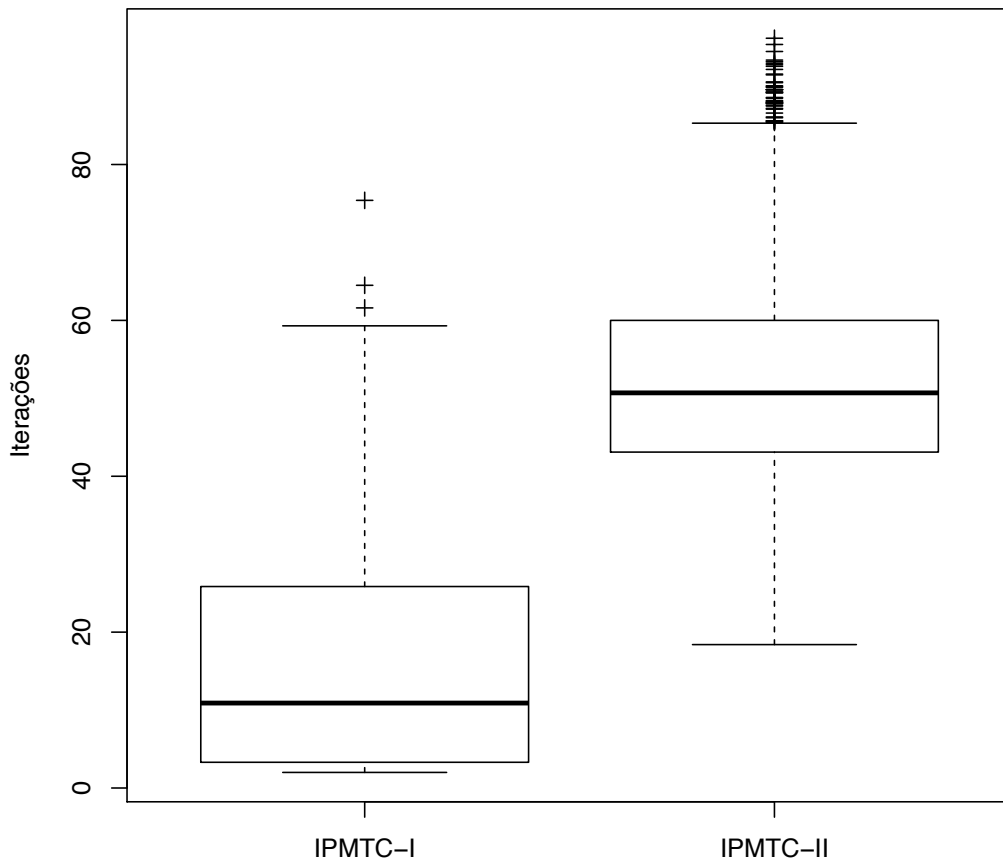
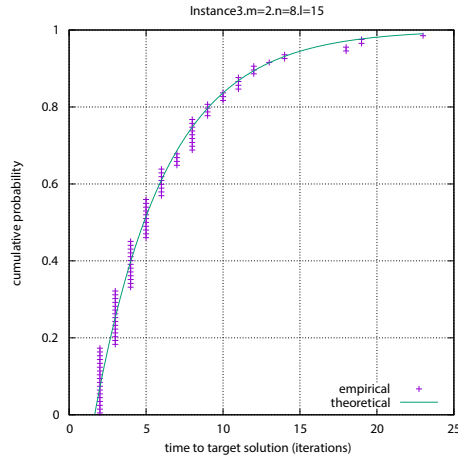


Figura 14 – Gráfico *boxplot* da análise de convergência média do BRKGA.

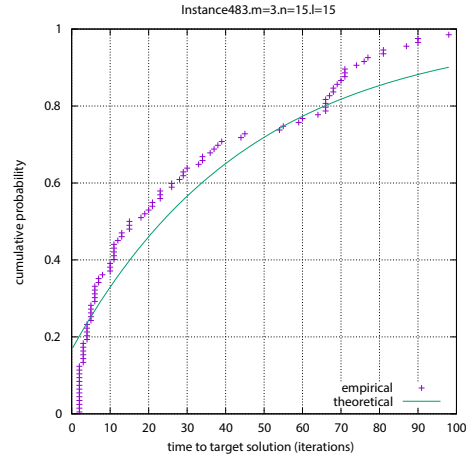
verde). Os gráficos ttt apresentam a probabilidade cumulativa (eixo y) de o método atingir uma solução alvo em cada número de iterações (eixo x).

A Figura 15 (a)-(f) ilustra a convergência do BRKGA para seis instâncias. As três primeiras instâncias, ilustradas em (a), (b) e (c), pertencem ao conjunto IPMTC-I. As três instâncias seguintes, ilustradas em (d), (e) e (f), pertencem ao conjunto IPMTC-II. No topo de cada gráfico, os valores para m , n e l são indicados. Todos os gráficos são interpretados de maneira análoga. Por exemplo, para a instância com oito tarefas, ilustrada em (a), é possível observar que a probabilidade do BRKGA encontrar uma solução tão boa quanto o valor alvo em 5 iterações é superior a 50%. Para a mesma instância, a probabilidade de encontrar a solução alvo em no máximo 15 iterações é maior que 90%.

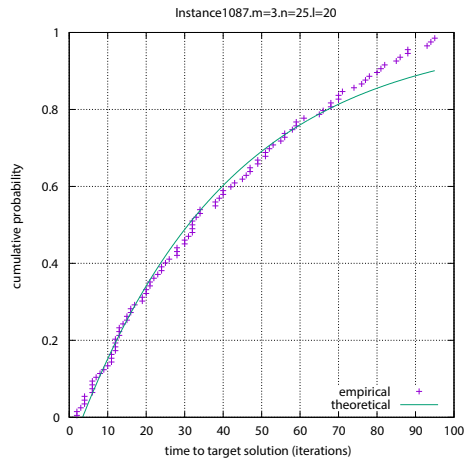
Conforme mencionado, os dados reportados pelos gráficos ttt ilustram o comportamento específico das instâncias selecionadas e não podem ser generalizados para todo o conjunto. Entretanto, conforme a convergência sumarizada no gráfico *boxplot*, é possível concluir que o número de iterações definido como critério de parada para o método



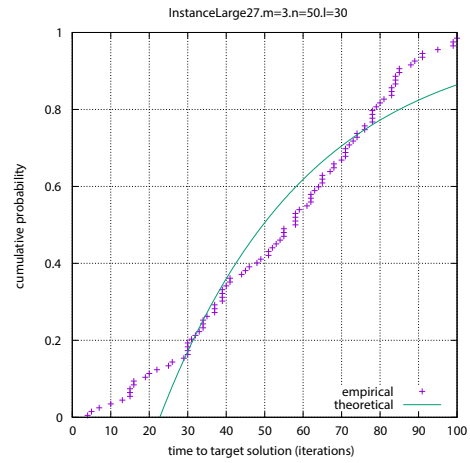
(a) Instância com 8 tarefas.



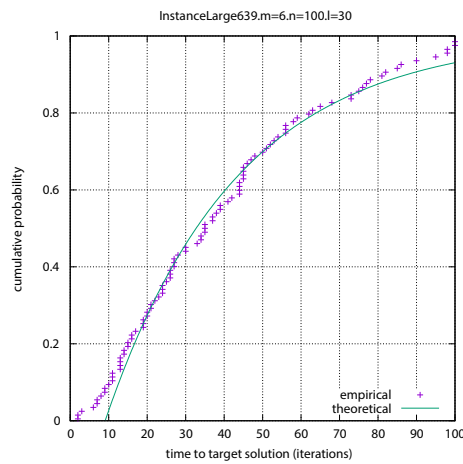
(b) Instância com 15 tarefas.



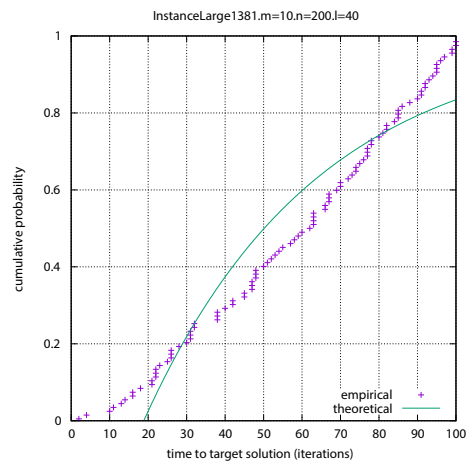
(c) Instância com 25 tarefas.



(d) Instância com 50 tarefas.



(e) Instância com 100 tarefas.



(f) Instância com 200 tarefas.

Figura 15 – Gráficos ttt ilustrativos para 6 instâncias.

proposto é suficiente para produzir soluções de alta qualidade para as instâncias de ambos os conjuntos.

6 Conclusão

Neste trabalho foi apresentada uma ampla pesquisa e o desenvolvimento de uma nova abordagem para o Problema de Escalonamento de Tarefas em Máquinas Flexíveis Paralelas com Restrições de Ferramentas (do inglês *Identical Parallel Machines With Tooling Constraints*, IPMTC), um problema NP-Difícil com diversas aplicações práticas no setor industrial. A pesquisa possibilitou: (i) a apresentação de uma revisão bibliográfica atual e abrangente para o problema tratado; (ii) a implementação do método que representa o estado da arte relacionado ao IPMTC, a metaheurística *Adaptive Large Neighborhood Search* (ALNS); (iii) a implementação de um modelo matemático para a resolução do problema; e (iv) a implementação e análise de um novo método para abordagem do IPMTC. A nova abordagem reportada consiste em uma implementação da metaheurística paralela Algoritmo Genético de Chaves Aleatórias Viciadas (ou BRKGA, do inglês *Biased Random-Key Genetic Algorithm*) hibridizado com três buscas locais aplicadas utilizando-se o método de Descida em Vizinhança Variável.

Os experimentos computacionais envolveram 2880 instâncias dos únicos dois conjuntos de instâncias *benchmark* disponíveis na literatura. O modelo matemático implementado foi utilizado para gerar soluções ótimas e limites superiores para as instâncias menores, pertencentes ao conjunto IPMTC-I. Infelizmente, para as instâncias maiores, pertencentes ao conjunto IPMTC-II, o modelo não foi capaz sequer de resolver a relaxação linear. A implementação do ALNS foi validada utilizando-se os resultados obtidos pelo método original. O *gap* médio de -3,91% e o tempo computacional médio 59,35% menor atestaram a qualidade superior da nova implementação.

Ignorando-se a divisão das instâncias em conjuntos, o método proposto apresenta 91,81% das melhores soluções, incluindo novas melhores soluções para 52,75% do total de instâncias e tempo de execução 92,69% menor. A análise estatística realizada comprovou que há diferença significativa entre os métodos comparados e que o método proposto é superior. Esta superioridade demonstrada pelo método proposto deve-se principalmente a implementação de operadores específicos para cada característica do IPMTC. O IPMTC agrega características do Problema de Escalonamento em Máquinas Paralelas e do Problema de Minimização de Trocas de Ferramentas, assim, as características de ambos problemas devem ser consideradas na resolução do IPMTC. O método proposto foi hibridizado com buscas locais específicas para as características dos subproblemas que compõem o IPMTC. Ao não negligenciar nenhuma característica do problema, o método proposto alcança resultados superiores ao ALNS, dominando o atual estado da arte para o IPMTC.

Os trabalhos futuros serão concentrados em áreas correlatas pouco exploradas na

literatura como abordagens considerando o desgaste de ferramentas, o compartilhamento de ferramentas entre máquinas, ambientes com máquinas não idênticas, abordagens multiobjetivo, o desenvolvimento de algoritmos exatos capazes de solucionar instâncias maiores do IPMTC e a geração de um novo *benchmark* com instâncias maiores, haja visto a facilidade para resolver as instâncias atuais com os recursos de paralelização disponíveis.

Referências

- 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 [33](#).
- 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 [77](#).
- 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 [31](#).
- 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 [31](#).
- 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 [31](#).
- 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 [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 [46](#).
- 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 12 vezes nas páginas [23](#), [29](#), [36](#), [37](#), [43](#), [67](#), [68](#), [69](#), [70](#), [72](#), [74](#) e [75](#).
- 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 [32](#).
- 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, p. 1–21, 2018. Citado 2 vezes nas páginas [29](#) e [31](#).
- CHAVES, A. A.; SENNE, E. L. F.; YANASSE, H. H. Uma nova heurística para o problema de minimização de trocas de ferramentas. *Gestão & Produção*, Universidade Federal de São Carlos, 2012. Citado na página [36](#).
- 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 5 vezes nas páginas [26](#), [29](#), [35](#), [43](#) e [53](#).

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 7 vezes nas páginas 35, 36, 37, 39, 41, 56 e 69.

GÖKGÜR, B.; HNIC, B.; ÖZPEYNIRCI, S. Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research*, Taylor & Francis, p. 1–17, 2018. Citado na página 37.

GONÇALVES, J. F.; MENDES, J. J. de M.; RESENDE, M. G. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, Elsevier, v. 167, n. 1, p. 77–95, 2005. Citado na página 29.

GONÇALVES, J. F.; RESENDE, M. G. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, Springer, v. 17, n. 5, p. 487–525, 2011. Citado na página 29.

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 2 vezes nas páginas 48 e 51.

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 2 vezes nas páginas 27 e 40.

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 2 vezes nas páginas 53 e 60.

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 33.

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 33.

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 2 vezes nas páginas 34 e 35.

LAPORTE, G.; SALAZAR-GONZALEZ, J. J.; SEMET, F. Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, Taylor & Francis, v. 36, n. 1, p. 37–45, 2004. Citado na página 37.

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 na página 71.

MCNAUGHTON, R. Scheduling with deadlines and loss functions. *Management Science, INFORMS*, v. 6, n. 1, p. 1–12, 1959. Citado na página 28.

MLADENović, N.; HANSEN, P. Variable neighborhood search. *Computers & operations research*, Elsevier, v. 24, n. 11, p. 1097–1100, 1997. Citado na página 62.

- MOHAMED, Z. M.; KUMAR, A.; MOTWANI, J. An improved part grouping model for minimizing makespan in fms. *European journal of operational research*, Elsevier, v. 116, n. 1, p. 171–182, 1999. Citado na página 34.
- 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 na página 60.
- PASSMARK. *CPU Benchmarks*. [S.l.], 2018. <http://www.cpubenchmark.net/cpu_list.php> Acessado em 10/10/2018. Citado 2 vezes nas páginas 70 e 74.
- PERSI, P. et al. A hierarchic approach to production planning and scheduling of a flexible manufacturing system. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 15, n. 5, p. 373–385, 1999. Citado 2 vezes nas páginas 34 e 35.
- PINEDO, M. *Scheduling: theory, and systems*. [S.l.]: Prentice-Hall, Upper Saddle River, NJ, USA, 2008. Citado 3 vezes nas páginas 27, 29 e 33.
- 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 2 vezes nas páginas 29 e 48.
- ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, Informs, v. 40, n. 4, p. 455–472, 2006. Citado na página 36.
- 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 73 e 75.
- SPEARS, W. M.; JONG, K. D. D. *On the virtues of parameterized uniform crossover*. [S.l.], 1995. Citado 2 vezes nas páginas 47 e 48.
- 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 28, 31, 32 e 54.
- STECKE, K. E.; SOLBERG, J. J. *The optimality of unbalanced workloads and machine group sizes for flexible manufacturing systems*. [S.l.: s.n.], 1982. Citado na página 31.
- STUDENT. The probable error of a mean. *Biometrika*, JSTOR, p. 1–25, 1908. Citado na página 75.
- TANG, C. S.; DENARDO, E. V. Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. *Operations research*, INFORMS, v. 36, n. 5, p. 767–777, 1988. Citado 5 vezes nas páginas 27, 37, 41, 43 e 52.
- 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 15, 48 e 72.
- VALENTE, J. M. et al. Genetic algorithms for single machine scheduling with quadratic earliness and tardiness costs. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 54, n. 1-4, p. 251–265, 2011. Citado na página 29.

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 33.

WOOD JR, T. Fordismo, toyotismo e volvismo: os caminhos da indústria em busca do tempo perdido. *Revista de administração de Empresas*, SciELO Brasil, v. 32, n. 4, p. 6–18, 1992. Citado na página 25.

YANASSE, H. H. Um novo limitante inferior para o problema de minimização de trocas de ferramentas. *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*, p. 2841–2848, 2009. Citado 2 vezes nas páginas 37 e 67.

ZEBALLOS, L. A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 26, n. 6, p. 725–743, 2010. Citado na página 25.