UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

PATRÍCIA MARQUES DE LIMA

HEURÍSTICA CONSTRUTIVA E GRASP PARA O SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS CNC COM HORIZONTE OPERACIONAL

BELO HORIZONTE
2023

PATRÍCIA MARQUES DE LIMA

HEURÍSTICA CONSTRUTIVA E GRASP PARA O SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS CNC COM HORIZONTE OPERACIONAL

Dissertação apresentada ao Curso de Engenharia de Produção no Programa de Pós-Graduação em Engenharia de Produção da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de mestre em Engenharia de Produção. Área de Concentração: Modelos e Algoritmos de Produção e de Redes

Orientador: Prof. Dr. Maurício Cardoso de Souza

BELO HORIZONTE

2023

Lima, Patrícia Marques de.
L732h Heurística construtiva e GRASP para o sequenciamento de tarefas em máquinas CNC com horizonte operacional / Patrícia Marques de Lima. – 2023.
1 recurso online (50 f. : il., color.) : pdf.

Orientador: Maurício Cardoso de Souza.

Dissertação (mestrado) – Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 49-50.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia de produção – Teses. 2. Máquinas – Monitoração – Teses. 3. Máquinas-ferramenta – Controle numérico – Programação – Teses. 4. Programação heurística – Teses. 5. Algoritmos de computador – Teses. I. Souza, Maurício Cardoso de. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 658.5(043)

Ficha catalográfica elaborada pelo Bibliotecário Marcio Anderson de Andrade Gomes CRB/6 2812
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia de Produção

FOLHA DE APROVAÇÃO

Heurística Construtiva e GRASP para o Sequenciamento de Tarefas em Máquinas CNC com Horizonte Operacional

PATRÍCIA MARQUES DE LIMA

Dissertação submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA DE PRODUÇÃO, como requisito para obtenção do grau de Mestre em ENGENHARIA DE PRODUÇÃO, área de concentração PESQUISA OPERACIONAL E INTERVENÇÃO EM SISTEMAS SOCIOTÉCNICOS, linha de pesquisa Modelos e Algoritmos de Otimização para Sistemas em Redes e de Produção.

Aprovada em 24 de maio de 2023, pela banca constituída pelos membros:

Prof(a). Mauricio Cardoso de Souza - Orientador (UFMG)

Prof(a). Horácio Hideki Yanasse (UNIFESP)

Prof(a). Joaquim José da Cunha Júnior (Afya Educacional)

Belo Horizonte, 24 de maio de 2023.

Documento assinado eletronicamente por **Mauricio Cardoso de Souza**, **Professor do Magistério Superior**, em 31/05/2023, às 15:04, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Joaquim José da Cunha Júnior**, **Usuário Externo**, em 31/05/2023, às 15:24, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Horacio Hideki Yanasse**, **Usuário Externo**, em 31/05/2023, às 15:46, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2322738** e o código CRC **1CF39433**.

Referência: Processo nº 23072.230829/2023-91 SEI nº 2322738

À minha família, por ser minha base. Ao meu companheiro, Pedro, por todo o apoio e suporte. Aos meus amigos, por toda a torcida e escuta em todos os momentos dessa jornada.

AGRADECIMENTOS

À orientação do Professor Doutor Maurício Cardoso de Souza, expresso minha profunda gratidão por guiar-me durante minha dissertação de mestrado. Sua paciência, apoio e ensinamentos foram inestimáveis.

Quero estender meus agradecimentos a todos os docentes com os quais tive a oportunidade de trabalhar. Sua contribuição proporcionou-me não apenas conhecimento intelectual, mas também a compreensão da importância do caráter e da afetividade na formação profissional.

Não posso deixar de reconhecer o apoio incondicional dos meus pais, irmã e companheiro. Durante os períodos em que me dediquei ao estudo universitário e estive ausente, eles sempre me fizeram compreender a importância da dedicação constante no presente para construir um futuro promissor. Seu apoio foi fundamental para que eu pudesse concluir esse ciclo.

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."

(Ariano Suassuna)

RESUMO

Máquinas de Comando Numérico Computadorizado (CNC) são controladas por comando de software e utilizam ferramentas dispostas numa caixa de ferramentas no processo produtivo. O número de ferramentas que a caixa comporta é geralmente inferior ao total necessário para a execução de todas as tarefas. É exigida então a parada do funcionamento da máquina para trocas de ferramentas, impactando no tempo de execução total das tarefas. Isso pode provocar o uso de horas extras e/ou atrasos nos prazos de entrega dessas tarefas. Nesse trabalho abordamos o problema de sequenciamento numa máquina CNC dentro de um horizonte de curto prazo, de 2 a 6 dias, propondo uma heurística GRASP para minimizar o custo com horas extras e atrasos. O algoritmo proposto foi testado em alguns cenários reais de chão de fábrica. Utilizamos as soluções fornecidas por um modelo de programação linear inteira, previamente proposto na literatura, como dados para mensurar a qualidade deste trabalho. Com isso, o algoritmo proposto mostrou ser uma alternativa promissora para obter boas soluções com esforço computacional reduzido.

Keywords: Heurísticas construtivas. GRASP. Sequenciamento de atividades. Troca de ferramentas.

ABSTRACT

Computer Numerical Control Machines (CNC) are controlled by tooling software and use tools arranged in a toolbox in the production process. The number of tools that the box holds is generally less than the total needed to carry out all the tasks. It is then the stop of the machine operation for tool changes, impacting the total execution time of the tasks. This may lead to overtime and/or delays in the delivery of these tasks. In this work, we address the problem of sequencing in a CNC machine short term horizon, from 2 to 6 days, proposing a GRASP heuristic to minimize the cost of overtime and delays. We used the solutions provided by an integer linear programming model, previously proposed in the literature, as data to measure the quality of this work. With that, the algorithm proposed proved to be a promising alternative to obtain good solutions with reduced computational effort.

Keywords: Constructive heuristics. GRASP. Machine scheduling. Tool switching.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CAD	Desenho Assistido por Computador
CAM	Manufatura Assistida por Computador
CNC	Comando Numérico Computadorizado
KTNS	*Keep Tool Needed Soonest*
MCHC	Ciclo Hamiltoniano de Custo Mínimo
MCHEA	Problema de Minimização de Custos de Horas Extras e Atrasos
PMTF	Problema de Minimização de Troca de Ferramentas
RCL	*Restricted Candidate List*

SUMÁRIO

1 INTRODUÇÃO

No presente trabalho abordamos um problema de sequenciamento de tarefas encontrado no uso de máquinas de Comando Numérico Computadorizado (CNC), utilizadas para a automação de processos e, com isso, melhoria do sistema de produção.

As máquinas CNC surgiram em 1946, na produção de hélices realizada por John Parsons, que criou um método de computação em cartão perfurado para calcular os pontos de uso de uma fresadora. Em 1958, John registrou uma patente para conectar o computador à máquina, o que ocorreu três meses antes do MIT, que trabalhava no mesmo conceito. No início dos anos 1970, no entanto, o próprio exército dos EUA popularizou o uso de computadores NC, construindo-os e alugando-os para vários fabricantes. O controlador CNC evoluiu em paralelo com o computador, gerando cada vez mais produtividade com automação nos processos de fabricação, especialmente na usinagem. Essas máquinas dependem de instruções digitais geralmente feitas em software de Manufatura Assistida por Computador (CAM) ou Desenho Assistido por Computador (CAD), então o programa de computador no controlador interpreta o projeto e realiza a execução do trabalho em maquinários como tornos, fresadoras, punsionadeiras, furadeiras etc.

Abordamos o caso de uma puncionadeira que realiza uma sequência de operações com um conjunto de ferramentas, localizadas em uma caixa de ferramentas, dada uma determinada programação do operador. O objetivo é executar diferentes tarefas que exigem diferentes ferramentas e, por normalmente não se ter capacidade suficiente para o uso simultâneo de todas as ferramentas nesta caixa, se torna necessário a parada do processo produtivo para reconfigurá-la. Dessa forma, parte relevante da perda de produtividade neste tipo de processo está associada ao tempo de parada para trocas de ferramentas (ou *setup time*). Além disso, a atividade de operação de máquinas envolve ao riscos de acidentes e ergonômicos como esforço físico intenso, levantamento de peso, mobiliário inadequado, controle rígido de tempo para garantir produtividade, imposição de ritmos excessivos, repetitividade e situações causadoras de estresse, que acarreta custos sociais e econômicos [MOREIRA 2015].

Seja $\mathcal{J}$ um conjunto de tarefas (ou *jobs*) a serem sequenciadas numa máquina CNC dentro de um horizonte composto por T períodos. Cada período corresponde a um dia de trabalho, composto por horas regulares e com a possibilidade de contratação de horas extras de dois tipos: 1 e 2. Para cada período t, considere u_t a sua capacidade de horas regulares, v_t^1 a sua capacidade de horas extras do tipo 1 e v_t^2 capacidade de horas extras do tipo 2. As horas

regulares não possuem custo atrelado e as horas extras do tipo 1 possuem um custo menor que as do tipo 2. Considere o_t^1 o custo de utilização de horas extras do tipo 1 e o_t^2 o custo de utilização de horas extras do tipo 2. Ao todo, são requeridas M ferramentas para executar todas as tarefas, e C $(C < M)$ é a capacidade da caixa de ferramentas da máquina, isto é, número máximo de ferramentas que podem ser carregadas simultaneamente.

Para cada um dos n jobs a serem executados, considere: (1) prazo de entrega, ou *due date* denominado por d_j; (2) tempo de execução denominado g_j; (3) multa diária de atraso denominada h_j. Nosso objetivo consiste em determinar uma sequência S de jobs e o período em que cada job da sequência será executado de forma que o custo total, considerando os custos com horas extras e multas de atraso, seja mínimo. Esse problema foi abordado por [da Cunha Júnior e de Souza 2010; da Cunha Júnior 2012] por meio de uma formulação de programação linear inteira.

O cenário que abordamos foi obtido em uma empresa de pequeno porte e conta com uma máquina puncionadeira que realiza furos e cortes em chapas que, apesar de ser assistida por comandos de software, têm um tempo valioso despendido para as atividades de setup. Para isso, consideramos que: (1) cada posição da caixa de ferramentas pode ser ocupada por uma única ferramenta, sendo ela de qualquer tipo; (2) não existem trocas simultâneas de ferramentas; (3) que o tempo de troca de uma ferramenta é o mesmo para qualquer ferramenta utilizada. Neste trabalho, nós propomos uma heurística construtiva e um algoritmo GRASP para o planejamento da sequência das tarefas a serem executadas em uma máquina CNC.

1.1 Objetivos

A seguir, descrevemos os objetvos gerais e específicos deste trabalho.

1.1.1 Objetivos Gerais

Compreender o problema de minimização de trocas de ferramentas em máquinas de manufatura flexível no âmbito industrial e utilizar as singularidades relacionadas ao negócio para embasar uma abordagem heurística.

Desenvolver uma abordagem eficiente para solucionar o problema aplicando a metaheurística GRASP para encontrar boas soluções para o problema em questão.

1.1.2 Objetivos Específicos

Realizar uma revisão da literatura sobre o problema de minimização de troca de ferramentas e também sobre GRASP.

Implementar o algoritmo GRASP com adaptações específicas para o problema em questão, tais como a estratégia de construção da solução inicial e avaliação de bons candidatos para seleção.

Realizar experimentos computacionais para avaliar a eficácia do algoritmo proposto em instâncias de diferentes tamanhos e com diferentes características, comparando os resultados obtidos com o modelo matemático proposto por [da Cunha Júnior e de Souza 2010; da Cunha Júnior 2012].

Concluir o trabalho apresentando os principais aprendizados e resultados obtidos, demonstrando as limitações e possibilidades de melhorias na abordagem proposta, sugerindo trabalhos futuros na área da pesquisa.

1.2 Organização do Trabalho

Após essa introdução, apresentamos brevemente os conceitos e trabalhos encontrados na literatura relacionados a este trabalho no Capítulo 2. No Capítulo 3, apresentamos os conceitos teóricos necessários para o entendimento deste trabalho, como a apresentação matemática da versão abordada do problema de sequênciamento: Problema de Minimização de Custos de Horas Extras e Atrasos (MCHEA). Bem como introduzimos conceitos necessários para o entendimento do método de resolução abordado.

No Capítulo 4, apresentamos o método GRASP desenvolvido, explicitando as etapas de construção e busca local. No Capítulo 5, apresentamos os resultados obtidos com o GRASP proposto, tendo como base para avaliar sua eficiência a execução num solver da formulação em Programação Inteira, proposta por [da Cunha Júnior e de Souza 2010; da Cunha Júnior 2012]. No Capítulo 6, avaliamos o tempo e a qualidade das soluções obtidas através dos experimentos computacionais, a partir a evolução das soluções encontradas por meio do método proposto e levantamos possibilidades de continuidade do trabalho apresentado.

2 REVISÃO DA LITERATURA

Neste capítulo, evidenciamos a fundamentação teórica deste trabalho, apresentando métodos exatos e métodos heurísticos encontrados na literatura para resolução do problema clássico de troca de ferramentas.

O Problema de Minimização de Troca de Ferramentas (PMTF) pode ser definido como segue: Seja $\mathcal{J}$ um conjunto de N jobs e M o número de ferramentas necessárias para executá-los. Cada job $j \in \mathcal{J}$ requer o uso de um subconjunto ferramentas, que devem estar presentes na caixa de ferramentas da máquina de manufatura para que haja a execução da tarefa. A caixa de ferramentas tem capacidade para uso de C ferramentas (com $C < M$). O objetivo é obter a sequência de execução dos jobs de forma a minimizar o número total de trocas de ferramentas.

Uma possível formulação do problema, proposta por [Tang e Denardo 1988] utiliza as variáveis de decisão x_{ij}, y_{ik} e z_{ik}, que são variáveis binárias definidas como:

$$x_{ij} = \begin{cases} 1 & \text{se na } i\text{-ésima posição do sequenciamento é executada a tarefa } j \\ 0 & \text{caso contrário} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{se na } i\text{-ésima tarefa processada a ferramenta } k \text{ está disponível} \\ 0 & \text{caso contrário} \end{cases}$$

$$z_{ik} = \begin{cases} 1 & \text{se na } (i-1)\text{-ésima tarefa processada a ferramenta} k \text{ não está disponível} \\ & \text{mas está disponível no processamento da } i\text{-ésima tarefa} \\ 0 & \text{caso contrário} \end{cases}$$

Considere uma matriz auxiliar A com $|\mathcal{J}|$ linhas e M colunas. Cada entrada a_{jk} desta matriz assume valor 1 se o job j utiliza a ferramenta m e 0 caso contrário. O modelo matemático é dado por:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{M} z_{ik}$$

$$\sum_{j \in \mathcal{J}} x_{ij} = 1 \quad i = 1, \ldots, n \tag{2.1}$$

$$\sum_{i=1}^{N} x_{ij} = 1 \quad \forall\, j \in \mathcal{J} \tag{2.2}$$

$$x_{ij} a_{jk} \leq y_{ik} \quad \forall\, j \in \mathcal{J}\ k = 1, \ldots, M\ i = 1, \ldots, N \tag{2.3}$$

$$\sum_{k=1}^{M} y_{ik} \leq C \quad i = 1, \ldots, N\ k = 1, \ldots, M \tag{2.4}$$

$$y_{ik} - y_{(i-1)k} \leq z_{ik} \quad i = 2, \ldots, N\ k = 1, \ldots, M \tag{2.5}$$

$$x_{ij}, y_{ik}, z_{ik} \in \{0,1\} \quad \forall\, j \in \mathcal{J}\ k = 1, \ldots, M\ i = 1, \ldots, N \tag{2.6}$$

Onde, a função objetivo tem como propósito minimizar a quantidade de ferramentas inseridas na máquina no início de cada tarefa, reduzindo, assim, a necessidade de trocas frequentes de ferramentas. As restrições 2.1 e 2.2 dizem respeito à distribuição de tarefas, exigindo que todas as tarefas sejam realizadas individualmente, isto é, uma tarefa só pode executada em uma única posição e uma posição só pode ser ocupada por uma única tarefa. Já as restrições 2.3 asseguram que todas as ferramentas necessárias para a execução de uma tarefa estejam disponíveis na máquina antes do início do seu processamento, enquanto as restrições 2.4 estabelecem que a capacidade C do compartimento de ferramentas deve ser respeitada. As trocas de ferramentas entre tarefas consecutivas são identificadas pelas restrições 2.5. Por fim, as restrições 2.6 referem-se aos valores possíveis que as variáveis podem assumir, garantindo a integralidade das variáveis de decisão.

2.1 Formulações e Limites Inferiores

O problema clássico de minimização de troca de ferramentas (PMTF) é conhecidamente NP-difícil (ver [Crama *et al.* 1994]). Entretanto, dada uma sequência de tarefas pré-definida, [Bard 1988] e [Tang e Denardo 1988] demonstraram que o número mínimo de trocas de ferramentas pode ser obtido em tempo polinomial. A proposta de [Tang e Denardo 1988] é a utilização de um modelo de programação linear inteira com a utilização de ferramentas as atividades representada em variáveis binárias. Alternativamente, [Laporte *et al.* 2004] propuseram uma formulação baseada no modelo para o problema do caixeiro viajante visto em [Dantzig *et al.* 1954], com uso de variáveis binárias para garantir que uma ferramenta necessária

na execução de uma tarefa esteja presente na máquina ao início de sua execução. Essa proposta domina em termos de relaxação linear a de [Tang e Denardo 1988], visto que esta é sempre zero. Já [Yanasse 2007] executou o cálculo de limites inferiores para o PMTF baseando-se na resolução de subproblemas, demonstrando que o valor ótimo do número mínimo de trocas de ferramentas para um subconjunto estrito do total de tarefas a serem executadas é menor ou igual ao valor ótimo do número mínimo de trocas necessárias para o processamento de todas as atividades. Dessa forma, seu procedimento consiste na seleção de um subconjunto de atividades, criando um subproblema cujo resolução é ótima. A partir daí, uma atividade é acrescentada por vez ao subproblema enquanto o esforço computacional para isso for considerado aceitável. [Yanasse *et al.* 2009] propuseram um cálculo de limites superiores ao valor ótimo do PMTF baseado em manter apenas as s melhores (s = 1, 2, 3, ...) sequências parciais em cada nível da árvore num esquema de enumeração.

[Ghiani *et al.* 2010] propuseram uma formulação do PMTF baseada no problema do Ciclo Hamiltoniano de Custo Mínimo (MCHC) com uma função objetivo não linear. Foi proposto um algoritmo de branch-and-cut para resolver esta formulação, e experimentos mostraram que ele poderia resolver problemas com até 45 jobs e 30 ferramentas em um tempo computacional relativamente curto comparado com a proposta de branch-and-bound de [Laporte *et al.* 2004].

[Burger *et al.* 2015] abordaram o Color Print Scheduling Problem, um problema da indústria onde a pintura de embalagens é feita com sobreposição de várias cores, onde as máquinas possuem um armazenamento limitado de cores o que exige paradas do processo para configurações da máquina e limpeza dos cartuchos de tinta, podendo ser modelado como PMTF. As formulações de PMTF de [Tang e Denardo 1988] e [Laporte *et al.* 2004] foram aplicados em um estudo de caso do mundo real. Os autores concluíram que o modelo de [Tang e Denardo 1988] teve melhor desempenho para instâncias pequenas e o modelo de [Laporte *et al.* 2004] teve melhor desempenho para instâncias maiores. Além disso, métodos de solução baseados na decomposição do PMTF em agrupamento de tarefas e sequenciamento de grupos, conforme proposto por [Salonen *et al.* 2006], foram implementados e comparados entre si.

Uma versão do PMTF foi considerada em [Adjiashvili *et al.* 2015]. Nesse estudo se manteve os tempos de troca de ferramentas, tempos de processamento dos jobs e mas considerou-se a capacidade de realizar trocas de ferramentas sem parar a máquina flexível. Isso é possível se houver espaços vazios suficientes na caixa. Este problema foi denominado de Machine Stopping Minimization e provou ser solucionável em tempo polinomial determinístico. Este

trabalho também introduziu uma nova função objetivo que minimiza o número de vezes que uma ferramenta é alocada em diferentes slots da caixa de ferramentas.

[Catanzaro *et al.* 2015] levantaram modelos de programação inteira para o PMTF, destacando as principais características de cada um e propuseram três novos modelos que obtiveram melhor limite de relaxação linear do que os modelos da literatura. Com os modelos propostos os autores foram capazes de resolver mais instâncias com o menor tempo de execução entre os modelos comparados. [Raduly-Baka e Nevalainen 2015] apresentaram o PMTF modular. Esta variante considera a existência de módulos de ferramentas que suportam conjuntos de ferramentas e podem ser trocados a um custo unário. Além disso, eles provaram que o PMTF modular também é um problema NP-Difícil.

2.2 Métodos heurísticos

Com relação à reduções e transformações para abordar o PMTF, [Crama *et al.* 1994] observaram que o problema pode ser transformado em um problema do caixeiro viajante, desde que todas as atividades demandem exatamente Q ferramentas (atividades são associadas a nós do grafo). Dessa forma, [Crama *et al.* 1994] propuseram algumas heurísticas baseadas em algoritmos para o problema do caixeiro viajante para serem aplicadas na existência de atividades que requerem menos de Q ferramentas. Tais heurísticas empregam diferentes estimativas ao número de trocas de ferramentas entre a execução de duas atividades, como distâncias entre dois nós no problema do caixeiro viajante. Similarmente, [Hertz *et al.* 1998] propuseram novas estimativas ao número de trocas de ferramentas como distâncias entre nós, e desenvolveram heurísticas para o PMTF fazendo uso de heurísticas refinadas para o problema do caixeiro viajante desenvolvidas por [Gendreau *et al.* 1992].

Uma versão do PMTF com ferramentas de tamanho não uniforme foi estudado por [Marvizadeh e Choobineh 2013]. O problema foi abordado com uma formulação de programação linear inteira, duas heurísticas de agrupamento de jobs e um algoritmo genético. Na maior parte dos casos, o algoritmo genético foi capaz de obter soluções melhores do que as heurísticas propostas em um tempo razoável em testes realizados com instâncias reais da indústria.

O método que representa o estado da arte para a resolução do PMTF atualmente é dado por uma metaheurística híbrida proposta por [Chaves *et al.* 2016]. Um procedimento é realizado para determinar regiões do espaço de busca com alta probabilidade de conter boas soluções. Um algoritmo genético de chave aleatória tendenciosa foi empregado como o compo-

nente metaheurístico de busca. Esses clusters promissores são então examinados intensivamente usando a variável vizinhança de descida, ou VNS do inglês variable neighborhood descent, como um procedimento de busca local. Este método, denominado CS+BRKGA, obteve recentemente os melhores resultados em muitos casos relatados na literatura, incluindo os propostos por [Yanasse *et al.* 2009] e [Crama *et al.* 1994].

Em trabalhos mais recentes, [da Cunha Júnior e de Souza 2009] abordaram o PMTF em uma indústria de pequeno porte do ramo metal-mecânica através de uma heurística gulosa para a obtenção de soluções viáveis que obtiveram ganhos significativos quando comparados às soluções praticadas na empresa onde o estudo foi conduzido. [Beezão *et al.* 2017] consideraram o PMTF em máquinas paralelas e propuseram uma implementação adaptativa de buscas de vizinhança e duas formulações matemáticas. Essas formulações tiveram dificuldades para resolver até mesmo instâncias pequenas com 15 jobs, e a metaheurística foi capaz de superar as duas heurísticas de [Fathi e Barnette 2002] em instâncias geradas aleatoriamente que representam seis diferentes cenários. Em [Furrer e Mütze 2017] foi apresentada uma estrutura baseada em branch-and-bound para o PMTF que considera tempos de configuração, processamento e trocas de ferramentas. Os experimentos computacionais usaram instâncias reais do problema Mailroom Insert Planning (problema de planejamento de inserção de correspondência) [Adjiashvili *et al.* 2015], que foram resolvidos de forma otimizada em poucos segundos.

3 PROBLEMA DE MINIMIZAÇÃO DE CUSTO DE HORAS EXTRAS E ATRASOS (MCHEA)

Neste capítulo, definimos formalmente o Problema de Minimização de Custo de Horas Extras e Atrasos (MCHEA). Apresentamos uma breve demonstração de sua complexidade computacional, a partir de resultados já conhecidos na literatura e sua formulação de Programação Linear Inteira.

3.1 Contexto industrial

O cenário que embasa esse estudo, em âmbito industrial, consiste na utilização de uma máquina puncionadeira com CNC. Esta máquina é centralizadora no processo de produção de uma industria mecânica de pequeno porte. O problema está diretamente ligado ao problema clássico de troca de ferramentas, uma vez que a máquina conta com uma caixa de ferramentas que tem capacidade inferior ao que necessita ser utilizado, demandando paradas na produção para trocas. Nesse caso prático, a máquina possui capacidade para carregar 10 ferramentas simultâneamente.

O planejamento da produção ocorre em horizonte semanal, podendo ser atualizado diariamente, tornando relevante a abordagem multi-período. Os custos do planejamento estão atrelados à utilização de horas extras e também à custos ocasionados por multas contratuais, geradas quando há atraso na entrega por não execução de um job dentro de seu prazo estabelecido.

Horas extras podem ser utilizadas para viabilizar a execução de todos os jobs demandados, entretanto é possível que jobs deixem de ser executados dentro do prazo ou pelo fato de não possuir mais horas extras para execução, ou por questões de custo-benefício, onde a multa atrelada ao atraso do job é inferior ao gasto com horas extras. Isto é, o job pode não ser executado dentro do horizonte de planejamento e, neste caso, a multa por atraso incide sobre o número de dias do horizonte menos o due date mais um.

O tempo necessário para realizar o ajuste da caixa de ferramentas depende da quantidade de ferramentas a serem trocadas. São realizadas diversas paradas no processo produtivo para configurar a caixa de ferramentas, baseando-se apenas no próximo job a ser executado. Portanto, há oportunidade de redução dos custos acarretados com essa atividade. Uma vez que se faça uma programação mais inteligente do uso das ferramentas, podemos diminuir essas interrupções para ajustes. É possível executar trocas de ferramentas ao final de um dia preparando a caixa de ferramentas para que o processamento de determinado job comece

no início do dia posterior, podem inclusive ser utilizadas horas-extras para tal. Assim, as trocas de ferramentas para processar o primeiro job de um período t, se necessário, podem ser feitas ao final do período t - 1 ou no início do período t.

O modelo apresentado a seguir foi elaborado por [da Cunha Júnior 2012], e incorpora todos os fatores descritos do contexto industrial. Esse é o panorama tratado no método GRASP proposto e experimentos executados, que serão abordados nos capítulos a seguir.

3.2 Formulação Matemática

Considere uma máquina CNC com capacidade para armazenar C ferramentas simultaneamente. Seja $\mathcal{J}$ o conjunto de jobs que desejamos sequenciar de forma a obter um planejamento da produção. Para execução dos jobs é necessária a utilização de M ferramentas ($M > C$). Considere que o planejamento possui um horizonte de T períodos, onde cada período corresponde a um dia de trabalho. Cada período pode ser dividido em N posições, sendo uma posição a representação da ordem de execução dos jobs no período, isto é, se um job é feito em $t = 2$ e $n = 3$, ele foi o terceiro job a ser executado no segundo dia de trabalho.

Para determinar o conjunto de ferramentas presente na caixa no início do período t utilizamos as variáveis binárias q_t^m, que assumem o valor 1 se a ferramenta m está presente na caixa no início do período t. Do mesmo modo, para verificar as trocas de ferramentas ao longo de um período, utilizamos as variáveis binárias w_{nt}^m, que assumem o valor 1 se a ferramenta m está na caixa para utilização na possição n do período t. E as variáveis binárias p_{nt}^m que assumem o valor 1 se a ferramenta m é adicionada à caixa para utilização depois da execução do job da posição n do período t. Essa forma de controle das variáveis nos ajuda a controlar a antecipação de setup da máquina quando se mostrar economicamente melhor.

A sequência de execução dos jobs é determinada com as variáveis binárias x_{nt}^j que assumem valor 1 se o job j é executado na posição n do período t. Para validar a execução utilizamos parâmetro a_m^j que indica se a tarefa j utiliza a ferramenta m. O tempo de processamento de cada tarefa é dado pelo parâmetro g_j e o tempo gasto com cada troca de ferramenta é o mesmo independente da ferramenta trocada, dado pelo parâmetro s.

O prazo para execução, ou due date, de um job j é dado por d_j e o custo por dia de atraso é de h_j. O atraso na execução de um job é determinado com as variáveis binárias y_k^j que assumem valor 1 se o job j ainda não foi executado no k-ésimo dia após seu due date. Isso quer dizer que se, em um horizonte de 7 dias de planejamento, um job j é executado no

dia 5 de planejamento quando deveria ter sido entregue até o dia 3, então $y_1^j = 1$, $y_2^j = 1$ e $y_3^j = y_4^j = y_5^j = 0$. Se algum job possuir o último dia de planejamento como dia de atraso, ele é considerado não executado.

Nosso objetivo consiste em minimizar uma função de custo composta por duas parcelas, uma relacionada às horas extras e outra relacionada aos atrasos de execução dos jobs. Na primeira, consideramos dois tipos de horas-extras: tipo 1 e tipo 2. A quantidade de utilização dessas horas por período t é representada, respectivamente, pelas variáveis v_t^1 e v_t^2. A essas horas extras são associados, respectivamente, custos unitários o_1 e o_2, sendo $o_1 < o_2$ e os limites de utilização dessas horas são dados por b_t^1 e b_t^2. A capacidade de produção em horas regulares de cada período t é dada por u_t. A segunda função de custo considera multas de atraso ou não execução dos jobs. Nesse caso, é possível assumir os custos de atraso e backlog. Assim, o modelo sempre tem solução viável, pois pode-se deixar de executar jobs quando não for possível (ou não oportuno) dentro do horizonte de planejamento.

Isso posto, podemos escrever o modelo de Minimização de Custos com Horas Extras e Atrasos (MCHEA) conforme proposto por [da Cunha Júnior 2012]:

$$\min \sum_{t=1}^{T}(o^1 v_t^1 + o^2 v_t^2) + \sum_{j\in\mathcal{J}}\sum_{k=1}^{T+1-d_j} h_j y_k^j$$

$$(\sum_{m=1}^{M}\sum_{n=1}^{N} p_{nt}^m + q_t^m)s + \sum_{j\in\mathcal{J}}\sum_{n=1}^{N} g_j x_{nt}^j \leq u_t + v_t^1 + v_t^2 \quad t = 1,\ldots,T \tag{3.1}$$

$$v_t^i \leq b_t^i \quad i = 1,2\ t = 1,\ldots,T \tag{3.2}$$

$$w_{(n+1)t}^m - w_{nt}^m \leq p_{nt}^m \quad i = 1,\ldots,N-1\ m = 1,\ldots,M\ t = 1,\ldots,T \tag{3.3}$$

$$w_{1t}^m \leq w_{N(t-1)}^m + p_{N(t-1)}^m + q_t^m \quad t = 2,\ldots,T\ m = 1,\ldots,M \tag{3.4}$$

$$w_{11}^m \leq q_1^m \quad m = 1,\ldots,M \tag{3.5}$$

$$a_m^j x_{nt}^j \leq w_{nt}^m \quad \forall\ j \in \mathcal{J}\ m = 1,\ldots,M\ n = 1,\ldots,N\ t = 1,\ldots,T \tag{3.6}$$

$$\sum_{m=1}^{M} w_{nt}^m \leq C \quad n = 1,\ldots,N\ t = 1,\ldots,T \tag{3.7}$$

$$\sum_{j\in\mathcal{J}} x_{1t}^j = 1 \quad t = 1,\ldots,T \tag{3.8}$$

$$\sum_{j\in\mathcal{J}} x_{(n+1)t}^j \leq \sum_{j\in\mathcal{J}} x_{nt}^j \quad n = 1,\ldots,N-1\ t = 1,\ldots,T \tag{3.9}$$

$$\sum_{n=1}^{N}\sum_{t=1}^{k} x_{nt}^j + y_{(k+1-d_j)}^j = 1 \quad \forall\ j \in \mathcal{J}\ k = d_j,\ldots,T \tag{3.10}$$

$$y_{(k+1)}^j \leq y_k^j \quad \forall\ j \in \mathcal{J}\ k = 1,\ldots,T+1-d_j \tag{3.11}$$

$$\sum_{k=T+2-d_j}^{T} y_k^j = 0 \quad \forall\ j \in \mathcal{J} \tag{3.12}$$

$$v_t^1, v_t^2 \geq 0 \quad t = 1,\ldots,T \tag{3.13}$$

$$x_{nt}^j, y_k^j, w_{nt}^m, p_{nt}^m, q_t^m \in \{0,1\} \quad \forall\ j \in \mathcal{J}\ m = 1,\ldots,M\ n = 1,\ldots,N\ t = 1,\ldots,T \tag{3.14}$$

A função objetivo do modelo minimiza o total dos custos, sendo a primeira parcela responsável pela contabilização dos custos de horas extras do tipo 1 e 2 e a segunda parcela pela contabilização dos custos de atraso. As restrições 3.1 garantem que o tempo total de execução dos jobs junto ao tempo de trocas de ferramentas não exceda a capacidade de processamento, em horas regulares e extras, para cada período. As restrições 3.2 garante que a realização de horas extras não exceda o limite. As restrições 3.3 controla a troca de ferramentas para que aquelas ferramentas necessárias numa posição $n+1$ e que não estão na máquina na posição n sejam inseridas, isto é, computadas na variável p. As ferramentas necessárias para execução do primeiro job do dia podem ser colocadas na caixa de ferramentas através de trocas ao final do período anterior ou, antes do seu processamento, no início do dia. Assim, as restrições 3.4

garantem que se um ferramenta requerida no primeiro instante de um determinado período não está na máquina no último instante do período anterior, ou ela deve ser adicionada através de uma troca ao final do período anterior ou através de uma troca no início do próprio período. As restrições 3.5 garantem que, para o período 1, as ferramentas requeridas ao processamento do primeiro job devem obrigatoriamente ser inseridas antes do seu processamento.

Todas as ferramentas necessárias para a execução de um job devem estar na caixa de ferramentas na posição em que esse é processado, conforme restrições 3.6. As restrições 3.7 garantem que o total de ferramentas presente na caixa de ferramentas não excede sua capacidade. As restrições 3.8 garantem que a primeira posição de cada dia seja ocupada por exatamente uma tarefa. Já as restrições 3.9 fazem o encadeamento das execuções, garantindo que uma posição $n+1$ só seja utilizada caso algum job tenha sido alocado a posição n.

As restrições 3.10 contabilizam os dias de atraso com relação à data máxima prevista para a entrega dos jobs. Enquanto as restrições 3.11 fazem o encadeamento da variável para contabilizar os dias de atraso corretamente. As restrições 3.12 atribuem zero aos atrasos maiores que o horizonte de planejamento, isto é, quando o job não é executado no horizonte de planejamento. As restrições 3.13 garantem a integralidade das variáveis que contabilizam as horas extras executadas e as restrições 3.14 garantem a integralidade das demais variáveis.

4 GRASP

Nesse capítulo, apresentaremos o método GRASP proposto para resolução do problema de minimização de custos de horas extras e atrasos no sequênciamento de jobs em máquinas CNC. Explicaremos as rotinas e a abordagem utilizada na construção do algoritmo, bem como detalhes sobre a construção da solução, vizinhanças e métodos de diversificação das soluções.

4.1 Contextualização

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma metaheurística que combina duas estratégias: uma heurística de construção gulosa e um procedimento de busca local aliados a aleatoriedade e adaptação, como o nome sugere. O algoritmo foi proposto por [Feo e Resende 1989] como uma metaheurística para resolução do problema de cobertura de conjuntos. Desde então, tem sido aplicado a uma ampla gama de problemas, incluindo coloração de grafos, caixeiro-viajante e roteamento de veículos.

A ideia básica do algoritmo GRASP é construir iterativamente uma solução viável adicionando elementos a uma solução parcial. A heurística de construção é tipicamente implementada usando uma versão aleatória de um algoritmo guloso. Os elementos a serem adicionados na solução são escolhidos aleatoriamente dentro de uma lista restrita de candidatos, chamada *Restricted Candidate List* (RCL), que contém os elementos mais promissores segundo um critério guloso. A RCL é composta de um subconjunto de elementos candidatos que são considerados para adição à solução parcial. Essa lista é gerada selecionando elementos com base em alguns critérios como custo, qualidade ou distância da solução atual. A RCL é a componente chave do algoritmo GRASP e seu tamanho determina o nível de aleatoriedade no algoritmo. Uma RCL pequena resulta em uma construção de solução mais próxima de um algoritmo guloso, enquanto uma RCL grande resulta em uma construção de solução mais aleatória.

Uma vez que uma solução viável é construída, o procedimento de busca local é usado para melhorá-la, fazendo algumas alterações na solução. O procedimento de busca local visa encontrar uma solução melhor na vizinhança da solução atual. Se uma solução melhor for encontrada, ela é aceita como a solução atual e o processo continua. Esse processo é realizado iterativamente até que um critério de parada seja satisfeito e o algoritmo finalize retornando a melhor solução encontrada até o momento.

O algoritmo GRASP pode ser resumido no pseudocódigo a seguir.

Algorithm 1 GRASP($itMax$, g, val)

```
1: it ← 0
2: bestVal ← +∞
3: bestSol ← ∅

4: while it < itMax do
5:     sol_c ← AlgoritmoConstrutivo(g)
6:     sol ← BuscaLocal(sol_c)
7:     if val(sol) < bestVal then
8:         bestVal ← val(sol)
9:         bestSol ← sol
10:    it ← it + 1
11: Return bestSol
```

O algoritmo inicia com um critério de parada definido, aqui representado por $itMax$, uma função g que determina seguindo um critério guloso, a pontuação dos nossos candidatos na RCL e uma função de avaliação da solução val. Enquanto o critério de término não for atendido uma solução é construida com uma heurística gulosa randomizada. Essa solução passa pelo procedimento de busca local para o refinamento da solução. A qualidade da solução é avaliada e, caso seja melhor, atualizamos a melhor solução. Esse processo é repetido até que o critério de parada seja atingido e a melhor solução encontrada é retornada.

[Festa e Resende 2002] realizaram vários estudos sobre a aplicação do GRASP a problemas de escalonamento. Em um estudo, eles aplicaram o GRASP ao problema de agendamento de enfermagem, onde o objetivo é designar turnos de enfermeiros de forma que os requisitos do hospital sejam atendidos e as preferências dos enfermeiros sejam levadas em consideração. Eles descobriram que o GRASP foi capaz de produzir soluções de alta qualidade em um período de tempo relativamente curto, superando outros métodos usados para comparação.

[Binato *et al.* 2002] aplicaram o GRASP ao job shop, que envolve o sequenciamento de um conjunto de jobs em um conjunto de máquinas. Eles compararam o GRASP com outros métodos e descobriram que o GRASP foi capaz de produzir soluções competitivas com as soluções mais conhecidas na literatura.

[Andrade e Resende 2007] propuseram uma nova abordagem baseada em GRASP com path-relinking para resolver o problema de escalonamento de migração de rede em telecomunicações. O problema envolve agendar a migração de serviços entre os elementos da rede, minimizando o impacto no desempenho da rede e no atendimento ao cliente. Os autores usam GRASP para gerar soluções de alta qualidade e melhorar o processo de busca combinando

soluções promissoras. A abordagem proposta é testada em instâncias reais de uma empresa de telecomunicações, e os resultados mostram que ela é capaz de produzir soluções que superam outros métodos utilizados para comparação.

O uso de metaheurísticas é normalmente motivado por problemas em que o espaço de busca é grande, a função de custo é complexa e a solução ótima não é facilmente previsível. Uma das principais vantagens do GRASP é que ele pode ser facilmente adaptado a domínios de problemas específicos por meio do ajuste fino de seleção gulosa e procedimentos de randomização. Também é relativamente simples de implementar e pode fornecer boas soluções em um período de tempo razoável, sendo uma escolha vantajosa em problemas de otimização do mundo real. Nesse estudo, buscamos explorar as potencialidades do algoritmo de construção concebido, inserindo a aleatoriedade no critério guloso para diversificar o espaço de busca, visando o ampliação custo-benefício da solução obtida.

4.2 Heurística construtiva

Propomos uma heurística construtiva que leva em conta aspectos como a similaridade dos jobs em termos do uso de ferramentas, os due dates, e os tempos de execução. Primeiro selecionamos aleatoriamente um número, possivelmente zero, de jobs para não serem executados no horizonte de planejamento. Em seguida, criamos uma sequência com os jobs a serem executados selecionando aleatoriamente um job por vez dentre uma lista restrita de candidatos (RCL). Esta RCL é composta por dois tipos de jobs ainda não sequenciados: (1) jobs que tenham maior similaridade com o estado atual da caixa de ferramentas e o menor due date; e (2) um número restrito de jobs de due date diferente do menor mas com menor tempo de execução. Por fim, realizamos um recorte temporal na sequência, determinando quais jobs da sequência são executados em cada período do horizonte de planejamento.

Nosso algoritmo construtivo utiliza uma lista ordenada auxiliar $L_{\bar{h}_j/g_j}$. Nessa lista os jobs são ordenados de forma crescente pela razão entre a multa de não execução do job e seu tempo de execução. A ideia é termos os jobs ordenados por benefício em se adicionar à sequência, logo os jobs iniciais são os menos interessantes e os jobs finais os mais interessantes em se executar. Além disso, o algoritmo conta com um conjunto de três parâmetros: $\bar{w}$ indica o número máximo de jobs que serão removidos da solução. Na etapa inicial do algoritmo construtivo, sorteamos um número w entre $\{0,\ldots,\bar{w}\}$ de jobs para remover. O parâmetro α indica o dimensionamento do espaço onde serão buscados esses w jobs. Esses jobs serão

sorteados entre os αw primeiros da lista $L_{\bar{h}_j/g_j}$. Por fim, o parâmetro ϕ dimensiona a busca de jobs similares com a máquina mas que não tenham chegado ao *due date*. Dessa forma, adicionamos uma espécie de perturbação com esse parâmetro, sorteando entre os ϕ primeiros jobs mais similares com a caixa de ferramentas que não estão vencendo, um job que pode entrar como candidato à sequência.

Seja J o conjunto dos jobs ainda não sequenciados, denominamos J_d o subconjunto de jobs não sequenciados com menor due-date, isto é, $J_d = \{j \in J : d_j \text{ seja mínimo}\}$. Seja Q_k o conjunto de ferramentas que estão na caixa (estado da caixa de ferramentas) antes de inserir o k-ésimo job na sequência e B_j o conjunto de ferramentas usadas por j, definimos por $|B_j \setminus Q_k|$ a similaridade de um job j em termos do uso de ferramentas, ou seja, quanto menos ferramentas sejam necessárias inserir na caixa para executar o job j maior é a similariade do job. Para montar a RCL consideramos os seguintes dois subconjuntos de J: $J'_d \subseteq J_d$ o subconjunto dos jobs de J_d que possuam maior similaridade com o estado corrente da caixa de ferramentas; e $J_x \subseteq (J \setminus J_d)$ um subconjunto de jobs com similaridade maior ou igual a dos jobs do conjunto J'_d (mas de due date maior) e com os menores tempos de execução.

No Algoritmo 2, a heurística de construção é iniciada na linha 1 com todos os jobs como passíveis de serem sequenciados. Tendo em vista que pode ser mais vantajoso em termos do custo total de uma solução pagar a multa $\bar{h}_j = h_j(T+1-d_j)$ de não executar um job j no horizonte de planejamento, consideramos $\bar{w}$ como um parâmetro que define o número máximo de jobs que podem ser não executados na solução a ser construída. Na linha 2 selecionamos aleatoriamente um número w entre 0 e $\bar{w}$ de jobs que não serão executados, e na linha 3 selecionamos aleatoriamente w jobs entre os αw primeiros elementos da lista $L_{\bar{h}_j/g_j}$ para não serem executados. Na lista $L_{\bar{h}_j/g_j}$ os jobs são ordenados de forma crescente pela razão entre a multa de não execução do job e seu tempo de execução. Na linha 4 atualizamos o conjunto de jobs a serem executados, removendo de J os w jobs selecionados e, nas linhas 5 e 6 inicializamos o custo da solução contabilizando as multas de atraso se houverem jobs não executados.

A partir da linha 7 até a linha 16, montamos uma sequência com os jobs de J. Consideramos o estado inicial da caixa de ferramentas na linha 7 e selecionamos os jobs de menor due date na linha 8. Para cada job a ser sequenciado, executamos o laço da linha 9 até a linha 16. Selecionamos o subconjunto de jobs com melhor similaridade com a caixa de ferramentas na linha 10. Na linha 11 selecionamos jobs com due date diferente que possuam similaridade melhor ou igual a detectada anteriormente. Na linha 12 os jobs de J_x são ordenados

Algorithm 2 Heurística de Construção ($\bar{w}$, α, ϕ)

Inicie J como o conjunto de todos os jobs
Selecione aleatoriamente em $\{0,1,\ldots,\bar{w}\}$ o número w de jobs que não serão executados
Seja $\bar{J}$ o conjunto de w jobs selecionados aleatoriamente entre os αw primeiros de $L_{\bar{h}_j/g_j}$
$J \leftarrow J \setminus \bar{J}$; $n_J \leftarrow |J|$
if $\bar{J} \neq \emptyset$ **then** $C \leftarrow \sum_{j\in\bar{J}} \bar{h}_j$
else $C \leftarrow 0$

Seja Q_1 o estado inicial da caixa de ferramentas
Seja $J_d \subseteq J$ o conjunto de jobs de J que tenham o menor due date
for $k = 1$ to n_J **do**
    Seja $J'_d = \{j \in J_d : |B_j \setminus Q_k|$ seja mínimo$\}$
    Seja $J_x = \{j \in J \setminus J_d : |B_j \setminus Q_k|$ seja menor ou igual ao observado em $J'_d\}$
    Caso $J_x \neq \emptyset$, ordene os elementos de J_x em ordem crescente de g_j e selecione aleatoriamente um job j^x entre ϕ primeiros
    Selecione um job aleatório j_k de $J'_d \cup \{j^x\}$
    $J \leftarrow J \setminus \{j_k\}$; Atualize J_d
    Remova $|B_{j_k} \setminus Q_k|$ ferramentas de Q_k tal que f_d^m seja mínima
    $Q_{k+1} \leftarrow Q_k \cup B_{j_k}$

Aplique KTNS na sequência gerada para obter os tempos de trocas de ferramentas
Seja $p_{j_k} = s_{j_k} + g_{j_k}$ onde s_{j_k} é o tempo de troca de ferramentas para execução do job j_k
Seja l_t o tempo de execução acumulado no período t
$t \leftarrow 1$; $l_1 \leftarrow 0$; $k \leftarrow 1$
while $t \leq T$ and $k \leq n_J$ **do**

    if $u_t - l_t \geq p_{j_k}$ **then** $co_{j_k} \leftarrow 0$
    else
        if $v_t^1 - l_t \geq p_{j_k}$ **then**
            if $l_t \leq u_t$ **then** $co_{j_k} \leftarrow o^1(l_t + p_{j_k} - u_t)$
            else $co_{j_k} \leftarrow o^1 p_{j_k}$
        else
            if $v_t^2 - l_t \geq p_{j_k}$ **then**
                if $l_t \leq u_t$ **then** $co_{j_k} \leftarrow o^1 v_t^1 + o^2(l_t + p_{j_k} - v_t^1)$
                else
                    if $l_t \leq v_t^1$ **then** $co_{j_k} \leftarrow o^1(v_t^1 - l_t) + o^2(l_t + p_{j_k} - v_t^1)$
                    else $co_{j_k} \leftarrow o^2 p_{j_k}$
            else $co_{j_k} \leftarrow \infty$

    if $\max\{0, h_{j_k}(t+1-d_{j_k})\} \geq co_{j_k}$ **then**
        $l_t \leftarrow l_t + p_{j_k}$; $C \leftarrow C + co_{j_k}$
        if $t > d_{j_k}$ **then** $C \leftarrow C + h_{j_k}(t - d_{j_k})$
    else
        if $u_t - l_t \geq s_{j_k}$ **then** $p_{j_k} \leftarrow g_{j_k}$
        $t \leftarrow t + 1$
        if $t > T$ **then**
            for $k' = k$ to n_J **do** $C \leftarrow C + \bar{h}_{j_{k'}}$
        else
            $l_t \leftarrow p_{j_k}$
            if $t > d_{j_k}$ **then** $C \leftarrow C + h_{j_k}(t - d_{j_k})$

    $k \leftarrow k + 1$

em ordem crescente de tempo de execução, se existirem tais jobs, e selecionamos um job aleatoriamente entre os ϕ primeiros. Na linha 13 selecionamos aleatoriamente o job para entrar na sequência. Na linha 14 atualizamos a lista de jobs a sequenciar e o subconjunto de jobs de menor due date. Denominamos de f_d^m a frequência da ferramenta m em J_d, isto é, o número de jobs de J_d que usam a ferramenta m. Na linha 15 fazemos, caso haja necessidade, as trocas de ferramentas para executar o job j_k removendo as ferramentas que possuem menor frequência. Por fim, na linha 16 atualizamos o estado da caixa de ferramentas.

As trocas de ferramentas que foram consideradas para se montar uma sequência, linhas 15 e 16, não necessariamente são as melhores possíveis depois que se tem a sequência montada. Assim, antes de realizar o recorte da sequência nos períodos do horizonte de planejamento, na linha 17 é aplicado o procedimento *Keep Tool Needed Soonest* (KTNS) à sequência gerada, para obtermos o mínimo de trocas de ferramentas. O procedimento consiste em, a partir do estado inicial da caixa Q_1, manter na caixa as ferramentas que serão usadas mais cedo na sequência (ver [Tang e Denardo 1988]). Com a aplicação do KTNS, temos o tempo s_{j_k} (possivelmente zero) de trocas de ferramentas, e consequentemente o tempo total p_{j_k} (setup mais execução) para executar o k-ésimo job da sequência na linha 18.

Para fazer o recorte temporal da sequencia, necessitamos de contabilizar o número l_t de horas executadas acumuladas no período t. Inicializamos l_t, t, e a posição na sequência na linha 20, e passamos a avaliar os períodos e tempo consumido com a execução da sequência, como pode ser visto nas linhas 21 a 45 do pseudocódigo. Para esse processo, iremos avaliar cada um dos jobs na sequência gerada e atribuímos a ele o período t para execução sempre que for possível fazê-lo sem consumo de horas extras, isto é, sem custo. Se, em determinada posição da sequência, observamos que um job requer utilização de horas extras para a conclusão de sua execução, iremos avaliar o custo-benefício em executá-lo no período t incorrendo em tal custo, ou executá-lo no período posterior.

O custo de hora extra co_{j_k} de se executar o job j_k no período t é calculado das linhas 22 a 33. Se for possível executar j_k sem hora extra co_{j_k} recebe zero na linha 22. Se forem necessárias horas extras dos tipos 1 e 2 isso é calculado das linhas 24 a 32. Se não for possível executar j_k no período t, pois neste caso as horas extras necessárias extrapolam os limites v_1^t e v_2^t, co_{j_k} recebe infinito na linha 33.

O custo-benefício de se executar o job j_k no período corrente t é avaliado na linha 34, dependendo da multa de executá-lo no período seguinte e o custo de execução no período corrente

considerando as horas extras. Se a multa for maior que o custo de execução, o job será executado no período corrente na linha 35, e na linha 36 verificamos se o job incorre em multa sendo executado no período corrente. Caso contrário, verificamos se há possibilidade de antecipar o setup de j_k na linha 38, isto é, se é possível fazer as trocas de ferramentas necessárias no período corrente sem incorrer em custos de horas extras, deixando apenas o tempo de execução do job propriamente dito para o período seguinte. Incrementamos o período corrente na linha 39 e caso tenhamos extrapolado o horizonte de planejamento fazendo $t = T + 1$, então contabilizamos as multas referentes à não execução dos jobs remanescentes da sequência na linha 41. Caso contrário, o job j_k é executado como o primeiro do novo período considerado, como visto na linha 43. Na linha 44 verificamos se o job incorre em multa sendo executado no novo período corrente, e finalmente incrementamos a posição da sequência na linha 45.

4.3 Busca Local

Propomos um procedimento de busca local para tentar melhorar as soluções geradas pelo algoritmo construtivo apresentado anteriormente. Nessa proposta, contamos com três diferentes movimentos que podem ser aplicados sobre um job para a geração das vizinhanças: exclusão, inserção e realoção. Verificamos cada um dos movimentos selecionando o melhor deles para ser executado a cada iteração da busca.

No movimento de exclusão selecionamos um job j para ser removido da sequência. Isto significa que o job não será executado, incorrendo no custo $\bar{h}_j$, e diminuindo a sequência em um elemento. Esse movimento é avaliado para cada job da sequência. No movimento de inserção selecionamos um job j que ao final da construção não foi executado para ser inserido na sequência, aumentando portanto a sequência em um elemento e economizando o custo $\bar{h}_j$. Esse movimento é avaliado para cada job não executado inserindo-o em cada posição da sequência. No movimento de realocação selecionamos um job que está numa posição k da sequência para ser movido para uma posição k' diferente de k. Cada movimento altera portanto a sequência e seu custo vai depender de como isso afeta as horas extras e eventuais multas. Para a avaliação do custo de cada movimento aplicamos à nova sequência gerada o procedimento KTNS e o recorte temporal visto anteriormente para a etapa construtiva (linhas 17 a 45 do Algoritmo 2).

4.4 Limitação de vizinhanças

Com o objetivo de tornar nosso GRASP mais eficiente, adicionamos limitantes ao explorar a vizinhança de uma solução para os movimentos de exclusão e realocação. Ao limitar o espaço de busca, nossa intenção é reduzir o número de soluções que precisam ser avaliadas, na esperança de reduzir o tempo computacional gasto na busca local. Isso é particularmente importante em problemas de otimização grandes ou complexos como o abordado, onde o espaço de busca pode ser extremamente grande acarretando tempos elevados de execução. Além disso, em um espaço de busca muito vasto podemos ter muitos ótimos locais e, com esses limites impostos, o algoritmo pode focar sua busca nas regiões mais promissoras do espaço. Evitando regiões que provavelmente não conterão boas soluções, podemos melhorar a qualidade das soluções encontradas aproximando-se do ótimo global.

No movimento de exclusão, utilizamos uma lista de jobs candidatos a não execução, previamente calculada, e que também foi usada na fase anterior (etapa construtiva) para avaliar a exclusão seletiva de candidatos com alto potencial de melhoria da solução corrente. Concentramos nossa atenção nos primeiros αw jobs da lista $L_{\bar{h}_j/g_j}$ (ver Seção 4.2) que já fazem parte da solução atual. Em outras palavras, examinamos a possibilidade de remover jobs que têm baixa relação entre a multa a ser paga em caso de atraso e o tempo de processamento requerido, isto é, que têm um bom potencial para melhorar o valor total da solução ao serem removidos da sequência.

No movimento de realocação, adicionamos algumas regras para mudar um job de posição na sequência, desde que este job esteja classificado entre os primeiros $\frac{|\mathcal{J}|}{3}$ jobs com maior tempo de processamento. A razão para isso é que esses jobs com maior tempo de processamento têm um impacto significativo na solução quando realocados, pois consomem uma grande quantidade de horas de execução durante o dia e, portanto, podem atrasar outros jobs e resultar em multas. O movimento de realocação de um job não é considerado nos seguintes casos: (i) se a multa do job está acima da mediana das multas e a nova posição após a realocação estiver depois do seu prazo de entrega e (ii) se a posição atual do job estava cumprindo seu prazo de entrega sem multa e a nova posição proposta é anterior à sua posição atual. Se alguma dessas duas condições for verdadeira, não realizamos a realocação porque acreditamos que essa mudança não resultaria em uma melhoria significativa na solução.

5 RESULTADOS COMPUTACIONAIS

Os resultados que apresentamos a seguir foram gerados num ambiente computacional Windows 10 de 64 bits com hardware equipado com processador intel Core TM i7 com clock de 2.6 GHz e 16GB de memória RAM. Implementamos os algoritmos na linguagem C++ versão 1.8 e utilizamos o CPLEX 12.1 como ferramenta para resolução dos problemas de programação linear inteira. Usamos como base de comparação neste experimento o modelo de programação inteira proposto por [da Cunha Júnior e de Souza 2010; da Cunha Júnior 2012], rodando o CPLEX com um limite de tempo de 7.200 segundos. Os métodos heurísticos foram implementados na linguagem C# em um ambiente Microsoft Visual Studio.

5.1 Instâncias

Nesse estudo utilizamos um conjunto de instâncias obtidos em um cenário de testes com dados reais e um conjunto de instâncias gerado aleatoriamente nos moldes dos dados reais. No conjunto de instâncias reais, os dados estão configurados com dois, quatro, e seis períodos que correspondem a dias de produção, com 10 instâncias para cada número de períodos. Cada instância tem um determinado número de jobs que requerem um certo total de ferramentas, estes valores são apresentados na Tabela 3. Consideramos uma máquina em que a caixa de ferramentas tem capacidade para 10 ferramentas, e o tempo de troca de uma ferramenta é 3 minutos. Os tempos de execução dos jobs variam entre 27 e 111 minutos. Os custos referentes a atrasos variam de acordo com a prioridade de atendimento do cliente, o valor por dia vai de 10 unidades monetárias para clientes pouco prioritários a 300 unidades monetárias para clientes prioritários. Os custos relacionados ao emprego de horas extras são de 55 unidades monetárias para a hora extra do tipo 1 e 70 unidades monetárias para a hora extra do tipo 2.

No conjunto de instâncias geradas aleatoriamente com distribuição uniforme, os dados estão configurados com quatro, e seis períodos que correspondem a dias de produção, com 20 instâncias para cada número de períodos. Cada instância tem um determinado número de jobs que requerem um certo total de ferramentas, o número de jobs foi sorteado em um intervalo de 27 a 36 para as instâncias de 4 períodos e 35 a 48 para instâncias de 6 períodos, que são os números de jobs mínimo e máximo das intâncias reais. Para o número de ferramentas diferentes utilizamos 34 e 35 ferramentas. Os valores são apresentados na Tabela 3. Consideramos uma máquina em que a caixa de ferramentas tem capacidade para 10 ferramentas, e o tempo de

troca de uma ferramenta é 3 minutos. Os tempos de execução dos jobs foram sorteados em um intervalo de 27 a 111 minutos, conforme os dados reais. Os custos referentes a atrasos variam de acordo com os dados reais, sendo sorteados em um intervalo de 10 unidades monetárias a 300 unidades monetárias. Os custos relacionados ao emprego de horas extras foi mantido em 55 unidades monetárias para a hora extra do tipo 1 e 70 unidades monetárias para a hora extra do tipo 2.

5.2 Desempenho do algoritmo construtivo multistart nas instâncias reais

O algoritmo construtivo foi testado numa versão multistart com diferentes combinações dos parâmetros $\bar{w}$, α, e ϕ, conforme descrito na Tabela 1. O número de iterações para analisar o algoritmo construtivo foi fixado em 2000. Na Tabela 1 apresentamos os gaps das melhores soluções obtidas e os tempos médios para cada combinação de parâmetros do algoritmo construtivo agrupando as instâncias por número de períodos. O gap é calculado em relação à solução obtida rodando o modelo matemático no CPLEX. Para obtenção do resultado médio desconsideramos *outliers* observados nas instâncias f6p2, f6p4 e f6p6, pois para estas instâncias todas as versões do algoritmo construtivo obtiveram soluções com gaps muito mais elevados do que os observados para as demais instâncias. Assim, tem-se as médias dos melhores gaps sobre 9 instâncias para cada grupo de 2, 4, e 6 períodos para cada versão do algoritmo construtivo.

É possível notar que todas as versões se beneficiaram de uma maior diversidade para a seleção do job de J_x a ser inserido na RCL, pois as versões com ϕ igual a 3 ou 5 têm resultados melhores do que as versões com ϕ igual a 1. O aumento do parâmetro α de 1 para 2 também leva a alguma melhoria, porém menos significativa. No entanto, o aumento de $\bar{w}$ acima de 2 piora ligeiramente os resultados do algoritmo construtivo.

Destacamos neste primeiro experimento a dificuldade em se obter soluções de alta qualidade para o problema. Mesmo executando 2.000 iteações de um algoritmo construtivo guloso randomizado os melhores gaps obtidos muitas vezes são bem elevados, principalmente para as maiores instâncias com 4 ou 6 períodos. Por exemplo, para a versão B3 que apresenta os menores gaps para as instâncias maiores, em 5 das 18 instâncias com $T = 4$ ou $T = 6$ o gap foi menor ou igual a 20%, e em 7 das 18 foi superior a 50%.

				$T=2$		$T=4$		$T=6$	
Versão	$\bar{w}$	α	ϕ	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)
A1	2	1	1	76,00	3	74,94	12	95,21	20
A2	2	1	3	31,65	3	56,87	12	69,96	20
A3	2	1	5	21,59	3	52,79	12	57,44	20
B1	2	2	1	73,20	3	74,94	12	95,21	21
B2	2	2	3	31,25	3	56,87	12	69,96	21
B3	2	2	5	21,37	3	52,79	12	57,44	21
C1	3	1	1	76,34	3	78,15	11	97,05	20
C2	3	1	3	26,30	3	65,33	11	66,65	20
C3	3	1	5	26,30	3	65,33	11	66,65	20
D1	3	2	1	74,12	3	78,15	11	97,05	20
D2	3	2	3	25,87	3	65,33	11	66,65	21
D3	3	2	5	23,34	3	56,09	11	65,15	20
E1	4	1	1	75,83	3	78,73	11	104,66	19
E2	4	1	3	27,95	3	62,89	11	69,03	20
E3	4	1	5	23,76	3	61,50	11	72,54	20
F1	4	2	1	75,45	3	78,73	11	104,66	20
F2	4	2	3	27,78	3	62,89	11	66,31	20
F3	4	2	5	23,42	3	61,50	11	72,45	20
G1	5	1	1	76,59	3	79,49	11	104,01	19
G2	5	1	3	31,71	3	67,32	11	74,26	19
G3	5	1	5	20,52	3	66,77	11	70,10	19
H1	5	2	1	74,01	3	79,49	11	104,01	19
H2	5	2	3	30,96	3	67,32	11	74,26	20
H3	5	2	5	20,52	3	66,77	11	70,10	20

Tabela 1 – Gaps e tempos médios obtidos para cada configuração do algoritmo construtivo.

Com o intuito de aprimorar a análise da influência dos parâmetros utilizados na obtenção dos gaps das soluções encontradas, apresentamos a seguir uma visualização mais clara por meio de gráficos gerados a partir dos dados da Tabela 1. Cada parâmetro utilizado na configuração do algoritmo resultou em um gráfico separado, onde os dados foram agrupados de acordo com a fixação dos demais parâmetros. Dessa forma, é possível observar de maneira mais detalhada a curva que representa a influência do parâmetro em destaque, considerando também o tamanho da instância, seja de 2, 4 ou 6 períodos.

Na Figura 1 foram agrupadas instâncias com os mesmos valores de α e Φ e para cada grupo o valor de $\bar{w}$ aumenta de 2 a 5. Por exemplo, para o grupo de instâncias A1, C1, E1, G1, tem-se $\alpha = 1$ e $\Phi = 1$ fixos, e $\bar{w} = 2$ para A1, $\bar{w} = 3$ para C1, $\bar{w} = 4$ para E1 e $\bar{w} = 5$ para G1. Observa-se que a tendência é que o gap aumente a medida que se aumenta o valor de $\bar{w}$ para α e Φ fixados. De maneira análoga, na Figura 2 foram agrupadas instâncias com os mesmos

valores de $\bar{w}$ e Φ, e na Figura 3 com os mesmos valores de $\bar{w}$ e α. Observa-se o algoritmo é pouco sensível ao valor de α e que gaps menores são obtidos com maiores valores de Φ.

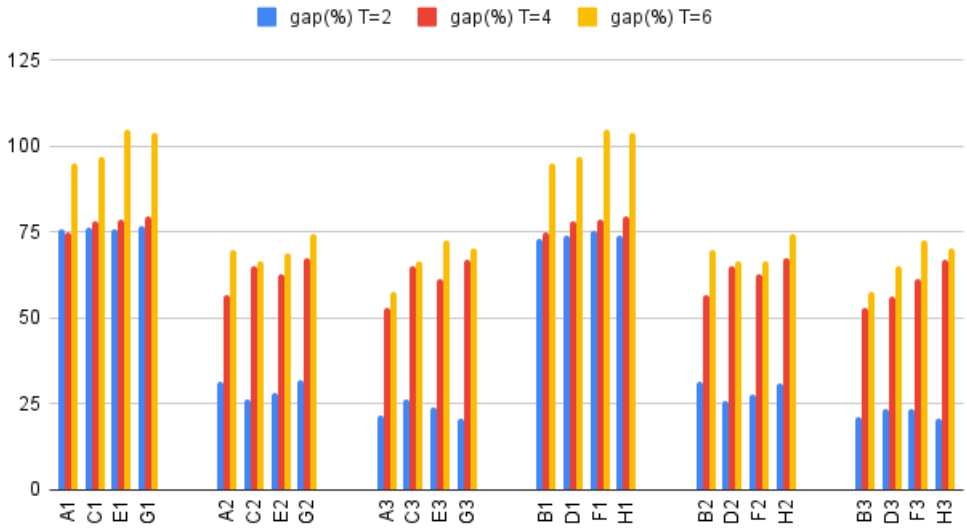

Figura 1 – Influência do parâmetro $\bar{w}$ nas soluções obtidas no algoritmo construtivo

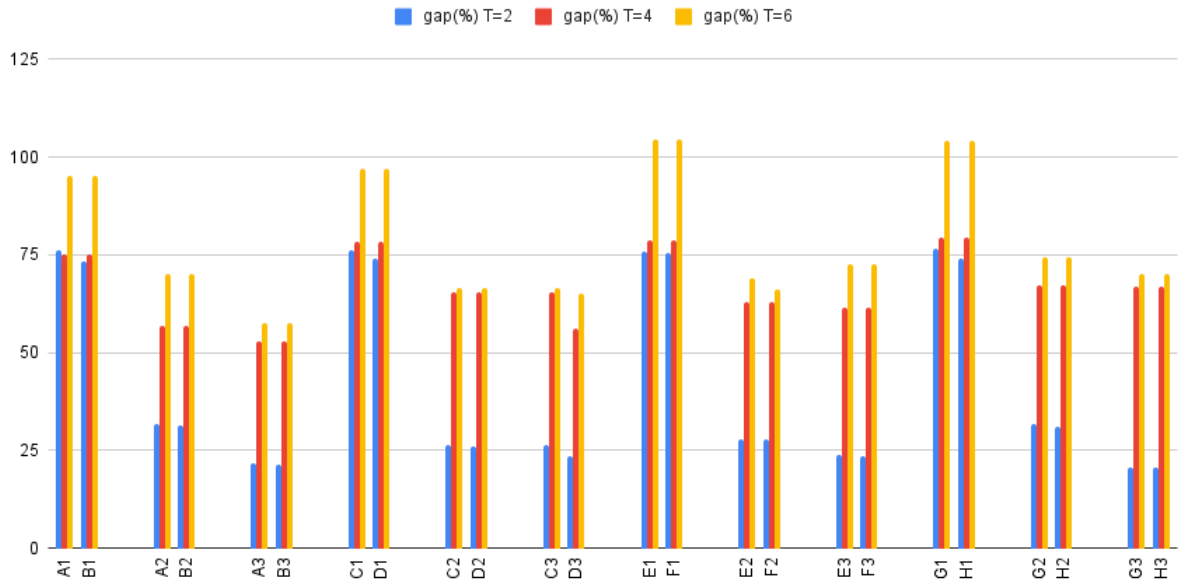

Figura 2 – Influência do parâmetro α nas soluções obtidas no algoritmo construtivo

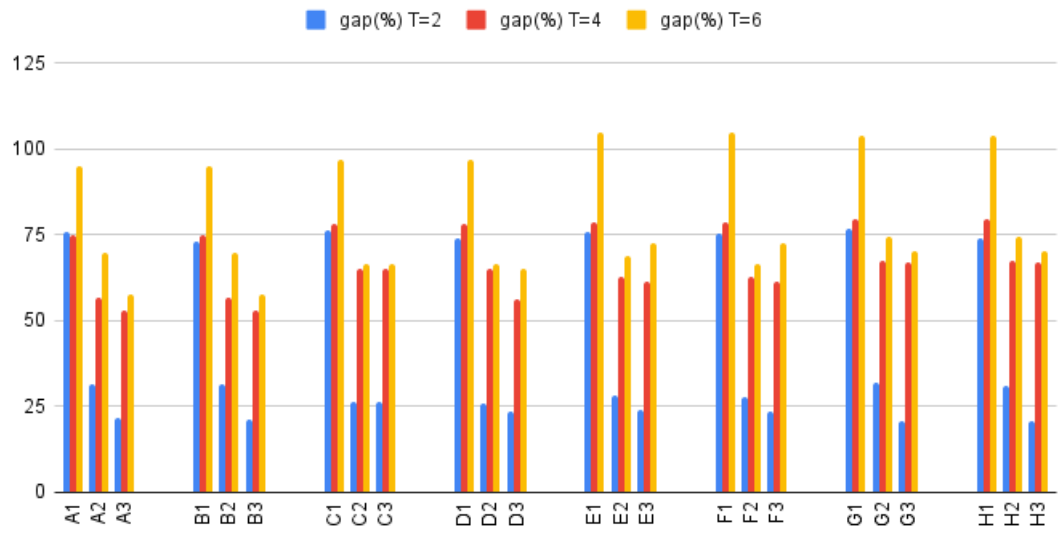

Figura 3 – Influência do parâmetro ϕ nas soluções obtidas no algoritmo construtivo

5.3 Desempenho do GRASP nas instâncias reais

A seguir, são apresentados os resultados obtidos por meio do algoritmo GRASP, utilizando o critério de *best improvement* para as atualizações da solução na etapa de busca local. São descritos os resulrados obtidos para as diferentes combinações de parâmetros, nomeadamente $\bar{w}$, α e ϕ. Para avaliar o desempenho do algoritmo, foi utilizado o melhor resultado obtido após a execução de 200 iterações, interrompendo o processo em caso de 50 iterações consecutivas sem melhoria da solução. Os resultados foram organizados na Tabela 2, onde são apresentados os valores de gaps e tempos médios para cada combinação de parâmetros, agrupando as instâncias de acordo com o número de períodos. É importante salientar que, para a avaliação do GRASP, o gap foi calculado em relação ao *lower bound* fornecido pelo CPLEX durante a resolução do modelo matemático executado em um limite de 2 horas.

				$T=2$		$T=4$		$T=6$	
Versão	$\bar{w}$	α	ϕ	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)
A1	2	1	1	9,18	6	35,33	148	43,80	774
A2	2	1	3	9,60	6	24,70	176	43,73	684
A3	2	1	5	8,89	7	33,68	236	43,39	684
B1	2	2	1	8,90	8	35,28	147	44,21	763
B2	2	2	3	9,60	6	24,70	176	43,68	725
B3	2	2	5	8,89	7	33,79	198	43,49	635
C1	3	1	1	8,82	5	35,04	170	43,80	740
C2	3	1	3	8,78	7	24,35	176	43,82	584
C3	3	1	5	8,40	7	34,12	170	43,88	603
D1	3	2	1	8,82	6	34,89	157	43,93	595
D2	3	2	3	8,78	8	24,30	158	43,42	745
D3	3	2	5	8,40	7	34,12	174	43,85	534
E1	4	1	1	8,82	5	34,78	146	43,76	659
E2	4	1	3	8,63	6	24,62	165	43,86	642
E3	4	1	5	8,40	7	33,80	168	43,76	557
F1	4	2	1	8,70	6	34,63	149	43,90	555
F2	4	2	3	8,39	6	24,33	172	43,99	698
F3	4	2	5	8,40	6	33,62	160	43,37	639
G1	5	1	1	8,89	5	34,87	125	44,41	599
G2	5	1	3	8,63	6	24,36	145	43,65	715
G3	5	1	5	8,89	6	34,56	144	44,01	612
H1	5	2	1	8,99	5	34,64	124	43,63	578
H2	5	2	3	8,31	6	24,23	175	43,62	724
H3	5	2	5	8,70	6	34,50	180	42,98	624

Tabela 2 – Gaps e tempos médios obtidos para cada configuração do algoritmo GRASP.

Com o intuito de aprimorar a análise da influência dos parâmetros utilizados na obtenção dos resultados do GRASP, apresentamos da Figura 4 a Figura 6 uma visualização mais clara por meio de gráficos gerados a partir dos dados da Tabela 2. Cada parâmetro utilizado na configuração do algoritmo resultou em um gráfico separado, onde os dados foram agrupados de acordo com a fixação dos demais parâmetros, de forma análoga a apresentada na seção anterior. Dessa forma, é possível observar de maneira mais detalhada a curva que representa a influência do parâmetro em destaque. Observa-se que com a busca local o algoritmo se beneficia com o aumento de $\bar{w}$ para determinadas configurações de α e Φ. Há uma mais nítida melhora do gap para $\alpha = 2$, e de modo geral melhores gaps são obtidos com $\Phi = 3$.

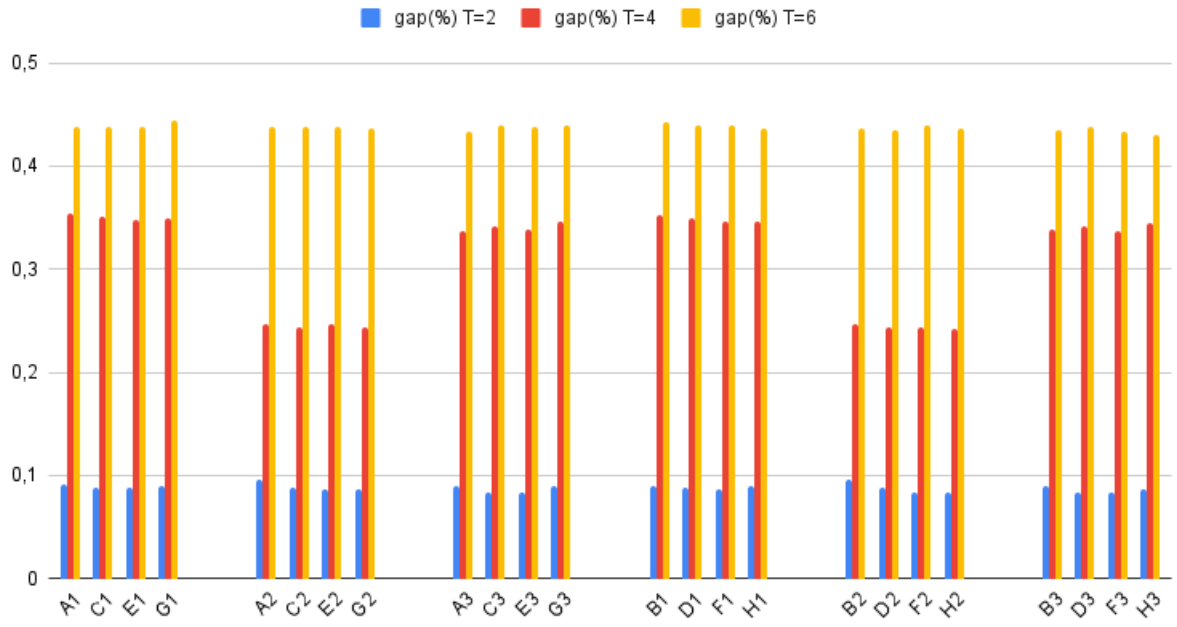

Figura 4 – Influência do parâmetro $\bar{w}$ nas soluções obtidas no GRASP

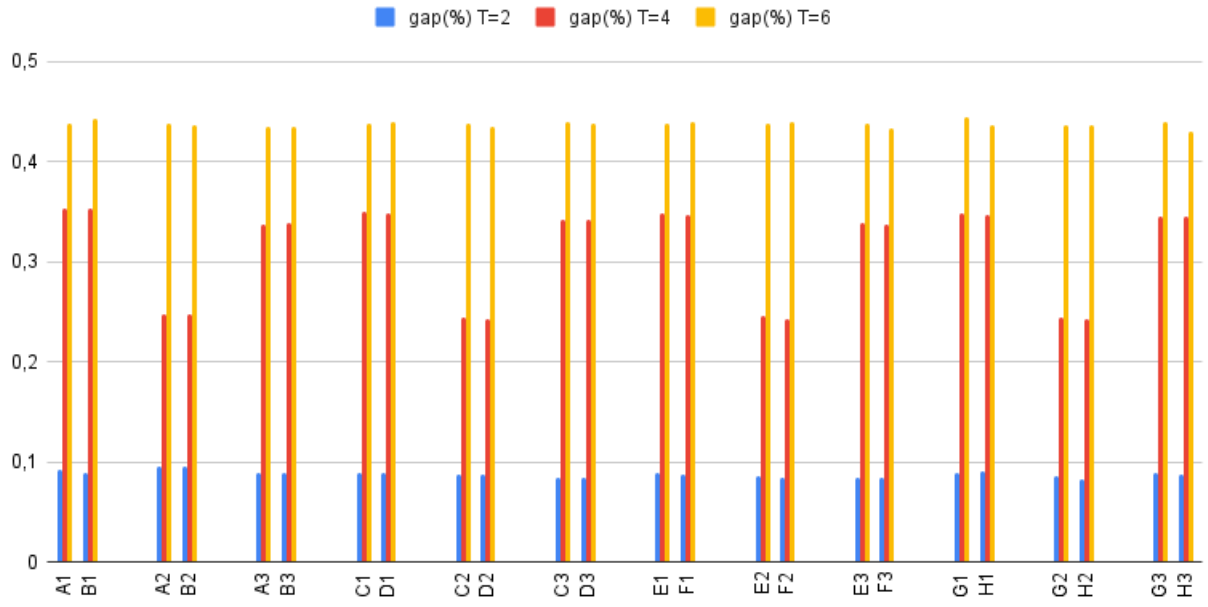

Figura 5 – Influência do parâmetro α nas soluções obtidas no GRASP

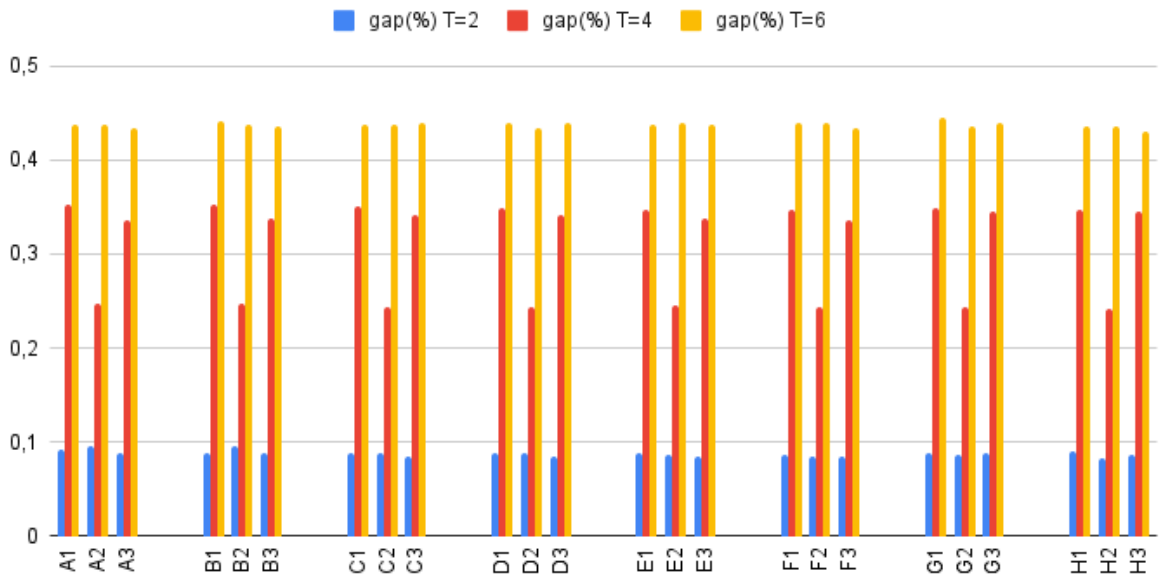

Figura 6 – Influência do parâmetro ϕ nas soluções obtidas no GRASP

Observamos que considerando a melhor parametrização do GRASP, H2, conseguimos obter resultados semelhantes e em alguns casos melhores, em termos de custo, do que o modelo matemático em tempo computacional reduzido. Na Tabela 3 indicamos para cada instância o número de jobs, o número total de ferramentas utilizadas pelos jobs da instância, e o número de períodos. Nas colunas dos resultados do GRASP apresentamos o gap calculado em relação ao *lower bound* obtido executando o modelo matemático no CPLEX, e o tempo de execução em segundos. Na coluna MIP apresentamos o gap de otimalidade fornecido pelo CPLEX. Na versão H2, conseguimos resultados iguais ou melhores em 14 das 30 instâncias testadas, sendo melhores em 9. Cabe notar que o tempo total para rodar o GRASP chegou no máximo a 25 minutos, sendo para a maioria das instâncias, bem abaixo disso. O modelo matemático atingiu o limite de tempo de 2 horas de execução sem obter o ótimo para 22 das 30 instâncias (cerca de 73%), e em muitos casos com gaps bem elevados, atestando mais uma vez a dificuldade do problema.

Instância				GRASP H2		MIP	
	n	M	T	gap(%)	t(s)	gap(%)	t(s)
f1p2	12	25	2	0,00	0	0,00	0
f1p4	35	34	4	21,81	330	20,72	7200
f1p6	47	35	6	24,18	1359	26,56	7200
f2p2	20	31	2	0,00	11	0,00	113
f2p4	36	35	4	9,73	555	9,72	7200
f2p6	46	35	6	11,19	1421	12,44	7200
f3p2	19	31	2	1,87	17	1,84	7200
f3p4	31	35	4	29,39	201	33,04	7200
f3p6	43	35	6	30,87	500	37,64	7200
f4p2	9	23	2	0,00	0	0,00	0
f4p4	27	34	4	50,00	22	50,00	7200
f4p6	35	34	6	69,65	110	69,64	7200
f5p2	17	33	2	9,35	4	10,12	7200
f5p4	31	34	4	26,02	99	26,00	7200
f5p6	42	35	6	42,48	435	44,70	7200
f6p2	14	27	2	0,00	0	0,00	0
f6p4	28	35	4	0,00	25	100,00	7200
f6p6	40	35	6	100,00	405	100,00	7200
f7p2	18	32	2	8,71	10	0,00	3261
f7p4	34	35	4	15,10	147	3,78	7200
f7p6	47	35	6	27,47	770	10,15	7200
f8p2	17	31	2	0,00	5	0,00	61
f8p4	28	34	4	33,10	50	33,10	7200
f8p6	42	35	6	64,26	251	64,26	7200
f9p2	18	31	2	0,00	12	0,00	2612
f9p4	36	35	4	16,12	223	8,82	7200
f9p6	48	35	6	14,86	1481	7,59	7200
f10p2	13	29	2	0,00	2	0,00	16
f10p4	31	35	4	41,00	97	47,33	7200
f10p6	40	35	6	51,27	503	54,26	7200

Tabela 3 – Comparativo entre GRASP com a parametrização H2 e modelo matemático em termos de GAP e tempo de execução.

5.4 Resultados das versões de vizinhanças limitadas nas instâncias reais

A seguir, apresentamos uma avaliação das diferentes versões da limitação da vizinhança de remoção (*L_REM*), da limitação da vizinhança de realoção (*L_REA*) e da combinação de ambas (*L_RR*). Essa avaliação foi realizada utilizando como base a parametrização $H2$ do algoritmo GRASP que apresentou os melhores resultados.

Observamos que ao adicionar a limitação da vizinhança, conseguimos obter algum ganho em termos de tempo de execução. Em média, o tempo de execução reduziu em 17%,

enquanto a qualidade da solução só piorou em mais de 1% do GAP em 4 das 30 instâncias testadas. Em apenas uma instância, f10p6, houve um aumento de gap maior que 2% em relação à solução H2 original.

Esses resultados mostram que o algoritmo proposto tem um grande potencial de escalabilidade e pode ser aplicado com sucesso em problemas mais complexos. Além disso, a adição da limitação da vizinhança pode melhorar significativamente o desempenho do algoritmo, tornando-o mais eficiente e econômico em termos de tempo de execução.

Instância				GRASP H2		H2 L_REM		H2 L_REA		H2 L_RR	
	n	M	T	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)
f1p4	35	34	4	0,00	0	0,00	0	0,00	0	0,00	0
f1p2	12	25	2	21,81	330	21,81	323	21,81	261	21,81	260
f1p6	47	35	6	24,18	1359	24,18	1344	25,84	579	25,84	557
f2p2	20	31	2	0,00	11	0,00	11	0,00	9	0,00	9
f2p4	36	35	4	9,73	555	9,73	551	10,99	407	10,99	389
f2p6	46	35	6	11,19	1421	11,19	1401	13,57	1175	13,57	1134
f3p2	19	31	2	1,87	17	1,87	16	1,87	14	1,87	14
f3p4	31	35	4	29,39	201	29,39	198	29,39	198	29,39	187
f3p6	43	35	6	30,87	500	30,87	492	30,87	398	30,87	381
f4p2	9	23	2	0,00	0	0,00	0	0,00	0	0,00	0
f4p4	27	34	4	50,00	22	50,00	21	50,00	16	50,00	16
f4p6	35	34	6	69,65	110	69,65	107	69,65	78	69,65	75
f5p2	17	33	2	9,35	4	9,35	4	9,35	4	9,35	3
f5p4	31	34	4	26,02	99	26,02	95	26,02	74	26,02	71
f5p6	42	35	6	42,48	435	42,48	427	42,48	510	42,48	489
f6p2	14	27	2	0,00	0	0,00	0	0,00	0	0,00	0
f6p4	28	35	4	0,00	25	0,00	24	0,00	18	0,00	17
f6p6	40	35	6	100,00	405	100,00	409	100,00	658	100,00	641
f7p2	18	32	2	8,71	10	8,71	10	8,71	12	8,71	12
f7p4	34	35	4	15,10	147	15,10	142	15,10	126	15,10	122
f7p6	47	35	6	27,47	770	27,47	756	23,88	887	23,88	844
f8p2	17	31	2	0,00	5	0,00	4	0,00	4	0,00	4
f8p4	28	34	4	33,10	50	33,10	49	33,10	43	33,10	41
f8p6	42	35	6	64,26	251	64,26	247	64,26	204	64,26	196
f9p2	18	31	2	0,00	12	0,00	12	0,00	10	0,00	10
f9p4	36	35	4	16,12	223	16,12	227	16,12	369	16,12	360
f9p6	48	35	6	14,86	1481	14,86	1461	14,86	1149	14,86	1094
f10p2	13	29	2	0,00	2	0,00	2	0,00	2	0,00	2
f10p4	31	35	4	41,00	97	41,00	95	41,00	94	41,00	89
f10p6	40	35	6	51,27	503	51,27	489	53,97	547	53,97	518
Média				23,28	302	23,28	297	23,43	262	23,43	251

Tabela 4 – Comparativo entre versões do GRASP H2 com vizinhanças limitadas, em termos de GAP e tempo de execução.

5.5 Avaliação de desempenho da versão paralelizada

Buscando melhorar o tempo de execução do algoritmo GRASP e aproveitar ao máximo os recursos computacionais disponíveis, realizamos a paralelização do algoritmo. A implementação do GRASP foi realizada na linguagem de programação *C#*, e os testes foram conduzidos em um sistema operacional Windows, equipado com um processador Intel Core i7 de 2.6 GHz, 16GB de memória RAM, 6 núcleos e 12 processadores.

Antes da paralelização, o algoritmo GRASP era executado em um loop for, onde eram realizadas até 200 iterações. Em cada iteração, um algoritmo construtivo era executado para construir uma solução inicial, seguido por uma busca local para melhorar essa solução.

Durante a paralelização, um processador mestre foi introduzido. Ele era responsável por ler os dados do problema, organizar as informações necessárias e distribuí-las para cada um dos processadores escravos. Essa distribuição foi realizada utilizando o método *Parallel.For* disponível na linguagem *C#*, o qual permite a execução paralela de um loop. O número máximo de iterações definido para o algoritmo GRASP durante a paralelização foi mantido em 200. Essas iterações foram igualmente divididas entre os processadores escravos disponíveis, garantindo uma carga de trabalho equilibrada entre eles. Nosso ambiente computacional conta com 12 processadores lógicos.

Os processadores escravos desempenhavam um papel fundamental na execução paralela do algoritmo GRASP. Cada um deles executava uma iteração do algoritmo construtivo, seguido pela aplicação da busca local na solução obtida. A melhor solução encontrada após essa busca local era então armazenada no pool de soluções compartilhado. Cada iteração era executada em uma thread independente. Cada processo escravo possui sua própria cópia do problema e a única informação compartilhada é o pool de soluções que deve ser atualizado com a melhor solução obtida. Após a conclusão do processamento realizado pelos processadores escravos, o processador mestre avalia todas as soluções presentes no pool e seleciona a melhor solução entre elas. Essa melhor solução é a solução final retornada pelo algoritmo GRASP paralelizado.

Em resumo, a paralelização do algoritmo GRASP buscou melhorar o desempenho ao executar as iterações do algoritmo construtivo e da busca local de forma simultânea em diferentes processadores. O uso de um processador mestre para coordenar as tarefas e o compartilhamento do pool de soluções foram estratégias adotadas para obter a melhor solução possível dentro de um tempo de execução razoável.

Apresentamos o comparativo entre as versões original, e das versões com limitações de vizinhança de forma sequencial e paralela na Tabela 5. Notamos uma redução do tempo médio de execução em cerca de 60%. Em instâncias maiores conseguimos alcançar reduções acima de 60% e apresentamos um tempo de execução melhor em 23 das 30 instâncias testadas. Nas outras 7 onde não se alcança o tempo do algoritmo sequencial isso se deve a interrupção do processo por 50 iterações sem melhoria da solução.

Instância				H2 ORIGINAL		H2 L_RR (S)		H2 L_RR (P)	
	n	M	T	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)
f1p4	35	34	4	0,00	0	0,00	0	0,00	1
f1p2	12	25	2	21,81	330	21,81	260	20,72	102
f1p6	47	35	6	24,18	1359	25,84	557	25,45	428
f2p2	20	31	2	0,00	11	0,00	9	0,00	9
f2p4	36	35	4	9,73	555	10,99	389	10,49	151
f2p6	46	35	6	11,19	1421	13,57	1134	11,02	427
f3p2	19	31	2	1,87	17	1,87	14	2,61	7
f3p4	31	35	4	29,39	201	29,39	187	28,96	68
f3p6	43	35	6	30,87	500	30,87	381	27,63	241
f4p2	9	23	2	0,00	0	0,00	0	0,00	0
f4p4	27	34	4	50,00	22	50,00	16	50,00	13
f4p6	35	34	6	69,65	110	69,65	75	69,65	67
f5p2	17	33	2	9,35	4	9,35	3	9,35	3
f5p4	31	34	4	26,02	99	26,02	71	26,02	50
f5p6	42	35	6	42,48	435	42,48	489	42,09	221
f6p2	14	27	2	0,00	0	0,00	0	0,00	1
f6p4	28	35	4	0,00	25	0,00	17	0,00	32
f6p6	40	35	6	100,00	405	100,00	641	100,00	157
f7p2	18	32	2	8,71	10	8,71	12	8,71	6
f7p4	34	35	4	15,10	147	15,10	122	15,10	74
f7p6	47	35	6	27,47	770	23,88	844	22,45	314
f8p2	17	31	2	0,00	5	0,00	4	0,00	4
f8p4	28	34	4	33,10	50	33,10	41	33,10	40
f8p6	42	35	6	64,26	251	64,26	196	62,61	174
f9p2	18	31	2	0,00	12	0,00	10	0,00	6
f9p4	36	35	4	16,12	223	16,12	360	16,12	161
f9p6	48	35	6	14,86	1481	14,86	1094	14,86	487
f10p2	13	29	2	0,00	2	0,00	2	0,00	2
f10p4	31	35	4	41,00	97	41,00	89	41,00	55
f10p6	40	35	6	51,27	503	53,97	518	53,10	180
Média				25,39	302	23,43	251	23,03	116

Tabela 5 – Comparativo entre versões do GRASP H2 com vizinhança limitada nas versões paralela e sequencial

5.5.1 Métricas

As métricas de desempenho de algoritmos paralelos são usadas para avaliar a eficiência e eficácia de algoritmos projetados para serem executados em sistemas de computação paralelos. Algumas das métricas de desempenho de algoritmos paralelos mais comuns são Speedup, Eficiência e Escalabilidade.

Speedup é a medida de quanto mais rápido um algoritmo pode ser executado em um sistema de computação paralela em comparação com um sistema de computação sequencial. É calculado como a razão entre o tempo gasto pelo algoritmo sequencial e o tempo gasto pelo algoritmo paralelo. Um aumento de velocidade de n significa que o algoritmo paralelo é n vezes mais rápido que o algoritmo sequencial. A eficiência é uma medida de quão bem um algoritmo paralelo utiliza os recursos de computação disponíveis. É calculado como a razão entre o aumento de velocidade alcançado pelo algoritmo paralelo e o número de processadores usados no sistema de computação paralela. A eficiência varia de 0 a 1, com 1 indicando que o algoritmo é perfeitamente eficiente, significando que o aumento de velocidade é linearmente proporcional ao número de processadores usados. Já Escalabilidade é a capacidade de um algoritmo paralelo de manter seu desempenho quando o tamanho do problema ou o número de processadores aumenta. Diz-se que um algoritmo é escalável se puder atingir um aumento de velocidade quase linear quando o número de processadores aumentar.

Focaremos na avaliação do *Speedup* e utilizaremos a Lei de Amdahl e a Lei de Gustafson-Barsis para mensurar o ganho em se paralelizar o algoritmo proposto. Essas leis tratam de dois modelos teóricos amplamente utilizados para analisar o desempenho de algoritmos paralelos. Em resumo, a Lei de Amdahl é útil para identificar o aumento de velocidade máximo teórico de um algoritmo paralelo, enquanto a Lei de Gustafson-Barsis é útil para medir o aumento de velocidade real que pode ser alcançado à medida que o tamanho do problema aumenta.

A Lei de Amdahl, em homenagem ao arquiteto de computadores Gene Amdahl, afirma que a aceleração de um algoritmo paralelo é limitada pela fração do programa que deve ser executada sequencialmente. Em outras palavras, se uma certa fração de um programa não pode ser paralelizada, então a aceleração do programa é limitada por essa fração. A fórmula da Lei de Amdahl é: $Speedup = 1/(F + ((1-F)/P))$, onde F é a fração do programa que deve ser executado sequencialmente, P é o número de processadores usados e *Speedup* é a razão entre o tempo necessário para executar o programa sequencialmente e o tempo necessário para executar o programa usando P processadores. A Lei de Amdahl é uma ferramenta útil para

determinar o aumento de velocidade máximo que pode ser alcançado por um algoritmo paralelo. [Hill e Marty 2008]

Por outro lado, a Lei de Gustafson-Barsis, em homenagem aos cientistas da computação John Gustafson e Edward Barsis, adota uma abordagem diferente para medir o desempenho do algoritmo paralelo. Em vez de assumir um tamanho de problema fixo, ele assume que o tamanho do problema aumentará à medida que o número de processadores aumentar. Em outras palavras, o algoritmo resolverá um problema maior na mesma quantidade de tempo. A fórmula da Lei de Gustafson-Barsis é $Speedup = P - F \times (P - 1)$, onde F é a fração do programa que deve ser executado sequencialmente, P é o número de processadores usados e *speedup* é a razão entre o tempo necessário para executar o programa sequencialmente e o tempo necessário para executar o programa usando P processadores. A Lei de Gustafson-Barsis sugere que, aumentando o tamanho do problema, o algoritmo paralelo pode atingir um aumento de velocidade linearmente proporcional ao número de processadores usados. [Gustafson 2011]

5.5.2 Avaliação

Para a execução do algoritmo, utilizamos um total de 12 processadores. Para avaliar a eficiência da versão paralelizada em relação à versão sequencial, utilizamos ambas as versões com as limitações de vizinhanças habilitadas. Para calcular o *speedup* do algoritmo paralelizado, utilizamos o tempo médio das execuções e nos baseamos nos resultados apresentados na Tabela 5. O *speedup* é obtido pela razão entre o tempo de execução sequencial e o tempo de execução em paralelo. Essa medida é importante para avaliar o desempenho do algoritmo paralelizado em relação à sua versão sequencial.

$$Speedup = \frac{251}{116} = 2,16$$

Para a avaliação dos índices de *speedup* teóricos, consideramos que 30% do algoritmo ainda é executado de forma sequencial, avaliando a leitura dos dados, pré processamento e montagem de estruturas de dados utilizados ao longo do algoritmo. Segundo a Lei de Amdahl, nosso *speedup* é dado por:

$$Speedup = \frac{1}{0,3 + \frac{0,7}{12}} = \frac{1}{0,358} = 2,79$$

Isso significa que temos como melhora máxima esperada da versão paralela uma melhoria na velocidade de execução de quase 3x com relação ao tempo sequencial. O que está alinhado com

as medições do experimento onde, em média, tivemos *speedup* de 2,16. O que significa uma redução de tempo para um pouco mais que metade do tempo gasto em média para a resolução das instâncias reais apresentadas.

Já segundo a Lei de Gustafson-Barsis, nosso *speedup* é dado por:

$$Speedup = 12 - 0,3 \times 11 = 8,7$$

esse índice teórico sugere o ganho estimado considerando um aumento do tamanho do problema.

A Lei de Gustafson-Barsis é uma abordagem complementar à Lei de Amdahl para estimar o speedup máximo que pode ser obtido em um algoritmo paralelo. Enquanto a Lei de Amdahl se concentra na proporção da parte sequencial do algoritmo para determinar o speedup máximo, a Lei de Gustafson-Barsis parte do tempo de execução paralela para estimar a eficiência máxima em relação ao tempo de execução sequencial. Essa lei sugere que, aumentando o número de processadores usados e o tamanho do problema a ser resolvido, a parte paralelizável do algoritmo pode crescer e o ganho de desempenho pode aumentar, chegando a um valor máximo estimado de 8,7 vezes em relação ao tempo de execução sequencial no nosso problema.

No entanto, é importante mencionar que a Lei de Gustafson-Barsis tem suas limitações. Por exemplo, ela assume que o tamanho do problema pode ser facilmente escalonado, o que nem sempre é possível. Além disso, a lei pressupõe que todos os processadores utilizados são igualmente eficientes, o que nem sempre é verdade. Além disso, a Lei de Gustafson-Barsis não considera possíveis gargalos no sistema de comunicação entre os processadores, que podem limitar o desempenho do algoritmo paralelo. Portanto, embora a Lei de Gustafson-Barsis seja útil para estimar o ganho de desempenho em um algoritmo paralelo, é importante considerar suas limitações e outras variáveis relevantes para uma análise mais precisa.

5.6 Desempenho do GRASP nas instâncias aleatórias

Apresentamos a seguir os resultados obtidos para o conjunto das instâncias aleatórias. Foi executado o modelo matemático, GRASP H2 Original, GRASP H2 com limitações de vizinhança sequencial (L_RR (S)) e paralizado (L_RR (P)), em 20 instâncias diferentes. Para cada instância, foram medidos o gap e o tempo de execução em segundos para ambos os métodos.

Analisando os resultados, observa-se que, em média, a heurística conseguiu reduzir o gap em cerca de 17%, em comparação com o *lower bound* obtido na execução do modelo matemático. Além disso, a heurística com reduções de vizinhança paralelizada foi significativamente

mais rápida, com tempo médio de execução de 214 segundos. É importante notar que em apenas 5 das 40 instâncias o modelo matemático obteve um resultado melhor que o GRASP, limitado no tempo de 2 horas de execução. O GRASP, por sua vez, obteve resultados significativamente melhores, com o algoritmo paralelizado executando em tempo máximo de 709 segundos e média de 214 segundos. Em 31 das 40 instâncias testadas houve redução no GAP obtido, mostrando consistência na melhoria das soluções como já observado nos testes executados nas instâncias reais.

	Modelo		H2 ORIGINAL		H2 L_RR (S)		H2 L_RR (P)	
Instância	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)	gap(%)	t(s)
c3_f1_p4	96,85	7200	94,90	334	95,08	210	95,10	155
c3_f1_p6	100	7200	0,00	55	0,00	18	0,00	244
c3_f2_p4	71,47	7200	50,15	748	52,09	614	49,21	188
c3_f2_p6	32,15	7200	38,37	671	41,48	332	40,33	226
c3_f3_p4	40,35	7200	38,69	121	38,69	97	38,69	67
c3_f3_p6	100	7200	100,00	245	100,00	181	100,00	161
c3_f4_p4	54,11	7200	42,88	371	43,90	249	36,19	192
c3_f4_p6	0	315	0,00	2	0,00	1	0,00	90
c3_f5_p4	58,21	7200	54,97	196	54,97	151	55,84	94
c3_f5_p6	87,36	7200	40,38	1286	44,81	793	40,35	573
c3_f6_p4	56,18	7200	51,40	113	51,34	123	50,98	75
c3_f6_p6	99,64	7200	65,65	986	66,04	734	66,04	365
c3_f7_p4	79,54	7200	57,40	528	57,40	376	57,40	143
c3_f7_p6	100	7200	100,00	1216	100,00	496	100,00	459
c3_f8_p4	42,17	7200	47,93	305	48,12	141	46,18	115
c3_f8_p6	0	3317	0,00	2	0,00	1	0,00	117
c3_f9_p4	29,42	7200	28,79	237	28,79	180	28,63	166
c3_f9_p6	47,61	7200	31,48	1045	31,02	596	31,94	353
c3_f10_p4	51,40	7200	49,17	171	52,06	115	51,73	104
c3_f10_p6	22,09	7200	27,40	216	27,40	159	26,23	136
c3_f11_p4	88,22	7200	57,58	434	61,47	168	57,46	116
c3_f11_p6	93,65	7200	45,29	1093	40,71	1507	41,22	709
c3_f12_p4	53,96	7200	46,43	186	46,43	146	45,98	126
c3_f12_p6	98,90	7200	49,21	542	49,53	405	49,28	223
c3_f13_p4	89,32	7200	42,39	298	42,26	363	40,97	200
c3_f13_p6	100	7200	0,00	3	0,00	19	0,00	95
c3_f14_p4	79,92	7200	49,04	564	52,60	251	51,03	176
c3_f14_p6	0	104	0,00	1	0,00	0	0,00	39
c3_f15_p4	49,91	7200	43,27	142	43,27	257	44,52	100
c3_f15_p6	100	7200	100,00	838	100,00	470	100,00	456
c3_f16_p4	90,97	7200	67,11	244	67,59	319	67,59	161
c3_f16_p6	92,07	7200	40,21	1558	43,25	1068	45,19	626
c3_f17_p4	53,97	7200	43,83	279	44,14	219	43,85	91
c3_f17_p6	98,60	7200	53,26	1581	58,43	691	57,46	516
c3_f18_p4	22,99	7200	30,76	184	30,16	135	31,35	76
c3_f18_p6	24,93	7200	32,66	384	20,98	257	18,49	171
c3_f19_p4	87,62	7200	82,55	283	82,81	249	82,35	206
c3_f19_p6	0	930	0,00	0	0,00	0	0,00	31
c3_f20_p4	56,19	7200	51,87	537	52,68	413	50,60	242
c3_f20_p6	100	7200	100,00	339	100,00	257	100,00	192
Média	63,74	6597	46,38	458	46,74	319	46,05	214

Tabela 6 – Comparativo entre a versão do GRASP H2 ORIGINAL, L_RR (sequencial) e L_RR (paralelizada), com o modelo matemático para as instâncias geradas aleatoriamente, em termos de GAP e tempo de execução

6 CONCLUSÃO

Neste capítulo, apresentamos brevemente as conclusões obtidas durante a elaboração deste trabalho, desde o conhecimento do cenário que motivou este estudo, juntamente à modelagem do problema, assim como a avaliação de esforço computacional e qualidade de solução obtida através das diferentes abordagens propostas. Apresentamos também possíveis novos trabalhos relacionados ao problema tratado.

Neste trabalho abordamos um problema prático de difícil resolução frequentemente encontrado em ambientes fabris que empregam máquinas de comando numérico. Em estudos anteriores, ver [da Cunha Júnior e de Souza 2010; da Cunha Júnior 2012], mostrou-se que os resultados obtidos com o uso de modelos matemáticos, embora na maioria dos casos não comprovadamente ótimos, geram ganhos expressivos na prática. Neste estudo, os resultados do GRASP proposto mostram que as alternativas heurísticas como método de solução podem apresentar ganhos significativos em tempo computacional reduzido, sendo uma linha de pesquisa promissora para abordar o problema.

Neste estudo, foi realizada uma análise detalhada das particularidades do problema, com o objetivo de identificar propriedades que possam ser exploradas para limitar as vizinhanças no processo de busca local, o que resultou em um algoritmo GRASP mais robusto e eficaz. Além disso, foi feita uma investigação sobre a paralelização do algoritmo, com a finalidade de melhorar a eficiência do tempo de execução e tornar a solução viável para problemas de planejamento com horizontes mais longos. Isso se mostra importante para ambientes industriais, nos quais é essencial uma programação antecipada, mas que também deve ser dinâmica para lidar com situações inesperadas. Assim, as melhorias propostas para o algoritmo GRASP pode trazer benefícios significativos para a resolução de problemas de planejamento em cenários reais.

Os resultados obtidos nesse estudo demonstram consistência na melhoria das soluções fornecidas pelo GRASP, já que em 31 das 40 instâncias aleatórias testadas houve redução no gap em comparação com o modelo matemático. Isso indica que o GRASP é uma heurística robusta e confiável para resolver o problema em questão. O algoritmo proposto apresentou um desempenho promissor para resolver as instâncias reais e aleatórias do problema, reduzindo significativamente o gap em relação ao modelo matemático e executando em um tempo muito mais rápido. Em termos de qualidade da solução, não foram observadas diferenças significativas entre as versões sequencial e paralela. O gap (diferença em relação à solução ótima conhecida) se manteve semelhante em todas as versões testadas. Os resultados mostram que a paralelização

do algoritmo GRASP foi eficaz em reduzir o tempo de execução, tornando-o mais eficiente e econômico em termos de tempo. Isso é especialmente benéfico em instâncias maiores, onde as reduções no tempo de execução foram mais acentuadas. A qualidade da solução não foi comprometida pela paralelização, pois o gap se manteve consistente em todas as versões testadas.

Para os trabalhos futuros pretendemos refinar o algoritmo construtivo considerando outros critérios para montar a RCL, e explorar algoritmos de vizinhança variável na fase de busca local do GRASP. Em termos de aplicação, pretendemos estender as heurísticas para tratar diferentes decisões de planejamento, como associação de jobs em produtos de manufatura compostos por subitens, o que atrela entre si due dates de subconjuntos de jobs, e equipamentos com mais de uma caixa de ferramentas.

REFERÊNCIAS

ADJIASHVILI, D.; BOSIO, S.; ZEMMER, K. Minimizing the number of switch instances on a flexible machine in polynomial time. **Operations Research Letters**, Elsevier, v. 43, p. 317–322, 2015.

ANDRADE, D.; RESENDE, M. Grasp with path-relinking for network migration scheduling. **Proceedings of International Network Optimization Conference**, (INOC 2007), p. 1–6, 2007.

BARD, J. A heuristic for minimizing the number of tool switches on a flexible machine. **IIE Transactions**, v. 20, p. 382–391, 1988.

BEEZÃO, A. C.; CORDEAU, J.-F.; LAPORTE, G.; YANASSE, H. H. Scheduling identical parallel machines with tooling constraints. **European Journal of Operational Research**, Elsevier, v. 257, p. 834–844, 2017.

BINATO, S.; HERY, W. J.; LOEWENSTERN, D. M.; RESENDE, M. G. C. A grasp for job shop scheduling. **Essays and Surveys in Metaheuristics**, Springer US, p. 59–79, 2002.

BURGER, A. P.; JACOBS, C.; VUUREN, J. H. van; VISAGIE, S. E. Scheduling multi-colour print jobs with sequence-dependent setup times. **Journal of Scheduling**, Springer, v. 18, p. 131–145, 2015.

CATANZARO, D.; GOUVEIA, L.; LABBÉ, M. Improved integer linear programming formulations for the job sequencing and tool switching problem. **European Journal of Operational Research**, Elsevier, v. 244, p. 766–777, 2015.

CHAVES, A. A.; LORENA, L. A. N.; SENNE, E. L. F.; RESENDE, M. G. Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. **Computers & Operations Research**, Elsevier, v. 67, p. 174–183, 2016.

CRAMA, Y.; KOLEN, A.; OERLEMANS, A.; SPIEKSMA, F. Minimizing the number of tool switches on a flexible machine. **International Journal of Flexible Manufacturing Systems**, v. 6, p. 33–54, 1994.

DA CUNHA JúNIOR, J. J. Programação de produção em máquinas CNC para o curto prazo. **Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Minas Gerais**, Dissertação de Mestrado, 2012.

DA CUNHA JúNIOR, J. J.; DE SOUZA, M. C. Minimização de trocas de ferramentas numa máquina cnc: aplicação de uma heurística gulosa a um caso real. **Anais do XLI Simpósio Brasileiro de Pesquisa Operacional**, 2009.

DA CUNHA JúNIOR, J. J.; DE SOUZA, M. C. Sequenciamento de tarefas em máquina de manufatura flexível para reduzir custos com horas extras e atrasos de entrega. **Anais do XLII Simpósio Brasileiro de Pesquisa Operacional**, p. 1201–1212, 2010.

DANTZIG, G.; FULKERSON, R.; JOHNSON, S. Solutions of a large-scale traveling-salesman problem. **Operations Research**, INFORMS, v. 2, p. 393–410, 1954.

FATHI, Y.; BARNETTE, K. Heuristic procedures for the parallel machine problem with tool switches. **International Journal of Production Research**, Taylor & Francis, v. 40, p. 151–164, 2002.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, v. 8, n. 2, p. 67–71, 1989.

FESTA, P.; RESENDE, M. G. Grasp: An annotated bibliography. **Essays and Surveys in Metaheuristics**, Springer US, p. 325–367, 2002.

FURRER, M.; MÜTZE, T. An algorithmic framework for tool switching problems with multiple objectives. **European Journal of Operational Research**, Elsevier, v. 259, p. 1003–1016, 2017.

GENDREAU, M.; HERTZ, A.; LAPORTE, G. New insertion and postoptimization procedures for the traveling salesman problem. **Operations Research**, v. 40, p. 1086–1094, 1992.

GHIANI, G.; GRIECO, A.; GUERRIERO, E. Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. **Networks: An International Journal**, Wiley Online Library, v. 55, p. 379–385, 2010.

GUSTAFSON, J. L. Gustafson's law. **Encyclopedia of Parallel Computing**, Springer US, p. 819–825, 2011.

HERTZ, A.; LAPORTE, G.; MITTAZ, M.; STECKE, K. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. **IIE Transactions**, v. 30, p. 689–694, 1998.

HILL, M. D.; MARTY, M. R. Amdahl's law in the multicore era. **Computer**, v. 41, n. 7, p. 33–38, 2008.

LAPORTE, G.; GONZáLEZ, J. J. S.; SEMET, F. Exact algorithms for the job sequencing and tool switching problem. **IIE Transactions**, v. 36, p. 37–45, 2004.

MARVIZADEH, S. Z.; CHOOBINEH, F. Reducing the number of setups for cnc punch presses. **Omega**, Elsevier, v. 41, p. 226–235, 2013.

MOREIRA, D. J. **O planejamento estratégico na prevenção de acidentes no trabalho: todos saem ganhando**. 2015. Disponível em: <https://jus.com.br/artigos/34194/o-planejamento-estrategico-na-prevencao-de-acidentes-no-trabalho-todos-saem-ganhando>.

RADULY-BAKA, C.; NEVALAINEN, O. S. The modular tool switching problem. **European Journal of Operational Research**, Elsevier, v. 242, p. 100–106, 2015.

SALONEN, K.; RADULY-BAKA, C.; NEVALAINEN, O. S. A note on the tool switching problem of a flexible machine. **Computers & Industrial Engineering**, Elsevier, v. 50, p. 458–465, 2006.

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, p. 767–777, 1988.

YANASSE, H. Um novo limitante inferior para o problema de minimização de trocas de ferramentas. **Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional**, p. 1886–1892, 2007.

YANASSE, H.; RODRIGUES, R.; SENNE, E. An enumeration algorithm based on partial ordering to solve the minimization of tool switches problem. **Gestão Produção**, v. 16, p. 370–381, 2009.