MATHEUS DE FREITAS ARAUJO

MÉTODOS EFICIENTES PARA O PROBLEMA DE *FLOW SHOP SCHEDULING* NÃO PERMUTACIONAL COM TRABALHADORES HETEROGÊNEOS

Dissertação apresentada a Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2017

Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa

T
A663m
2017

Araujo, Matheus de Freitas, 1991-
Métodos eficientes para o problema de flow shop scheduling não permutacional com trabalhadores heterogêneos / Matheus de Freitas Araujo. – Viçosa, MG, 2017.
ix, 60f. ; 29 cm.

Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Inclui bibliografia.

1. Algoritmos. 2. Otimização combinatória. 3. Heurística.
I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação.
II. Título.

CDD 22 ed. 005.1

MATHEUS DE FREITAS ARAÚJO

MÉTODOS EFICIENTES PARA O PROBLEMA DE FLOW SHOP SCHEDULING NÃO PERMUTACIONAL COM TRABALHADORES HETEROGÊNEOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 07 de março de 2017.

Mauro Nacif Rocha

Luciana Brugiolo Gonçalves

José Elias Claudio Arroyo
Orientador

Dedico este trabalho ao meus pais, Leda e Carlos, ao meu irmão Marcel, a Marcela Sotta e à todos meus amigos.

Agradecimentos

Agradeço a Deus por me amparar nos momentos difíceis, me dar força interior para superar as dificuldades e por sempre estar ao meu lado.

À minha Mãe Leda e ao meu Pai Carlos, por sempre estar ao meu lado com o seu amor eterno e incondicional. Obrigado por terem me educado, pelos conselhos, incentivos, afeto, amizade, dedicação e por não medirem esforços para tornar meus sonhos possíveis. Se hoje cheguei aqui, tenho muito a agradecê-los. Obrigado ao meu irmão Marcel pela cumplicidade e amizade.

À minha família, a qual amo muito, pelo carinho, paciência e incentivo.

À Universidade Federal de Viçosa pela oportunidade de realizar o curso. Ao Professor José Elias, meu orientador, por sua dedicação, paciência, confiança e por seus ensinamentos. Agradeço também a todos os docentes do Departamento de Informática da UFV que de alguma forma contribuíram para o meu crescimento intelectual e profissional. Aos funcionários do departamento, pelas diversas assistências e agradáveis momentos de distração.

Aos amigos que fizeram parte dessa caminhada, sempre me ajudando e incentivando a concluir o programa. Obrigado José Cleydson, Thales, Renan Oliveira, Rubens, Webert, Matheus e Luiz Augusto sem vocês esse sonho não seria possível. Obrigado todos os colegas do mestrado pelo companheirismo.

Ao Grupo BMS, em especial a família Sabino, pela compreensão e paciência nos momentos difíceis.

Ao José Luiz pelos conselhos e orientações espirituais

À Marcela Sotta. Sua ajuda e apoio foram para mim de valor inestimável, e qualquer palavra que eu utilize seria insuficiente para agradecê-la com o devido

merecimento. Obrigado por acreditar em mim quando eu achei difícil acreditar em mim mesmo. Tenho certeza que sem você eu não teria chegado até aqui.

Por fim, o meu profundo e sentido agradecimento a todas as pessoas que contribuíram para a concretização desta dissertação, estimulando-me intelectualmente e emocionalmente.

"Você não consegue ligar os pontos olhando pra frente; você só consegue ligá-los olhando pra trás. Então você tem que confiar que os pontos se ligarão algum dia no futuro."Steve Jobs

Sumário

Resumo

ARAUJO, Matheus de Freitas, M.Sc., Universidade Federal de Viçosa, março de 2017. **Métodos eficientes para o problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos**. Orientador: José Elias Claudio Arroyo.

Este trabalho aborda o problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH). Sendo este classificado como um problema multicomponente, uma vez que, combina dois problemas no qual o resultado de um afeta o outro. O FSNPTH é composto por dois problemas clássicos de otimização combinatória: alocação de trabalhadores em máquinas e o *flow shop scheduling* não permutacional (FSNP). O problema consiste em alocar trabalhadores heterogêneos em máquinas dispostas em série, na qual a heterogeneidade se dá pelo tempo gasto pelo trabalhador ao operar uma máquina. A alocação dos trabalhadores definem os tempos de execução das tarefas do problema de FSNP. O objetivo do problema de FSNPTH é minimizar o tempo máximo de conclusão das tarefas, conhecido como *makespan*. Para resolve-lo, inicialmente é proposto a aplicação do método *Proximity Search* (PS) para tentar determinar soluções ótimas para o problema utilizando o modelo de programação linear inteira mista 0-1 (PLIM). Esse método consiste em substituir a função objetivo por uma função de proximidade e adicionar uma restrição de corte no modelo. Iterativamente o novo modelo é resolvido e a restrição de corte é atualizada. Isso garante que PS limite o espaço de busca e identifique as soluções ótimas. Foram desenvolvidas três versões do PS denotadas por PS_1, PS_2 e PS_{2RINS}. Dado que o problema pertence à classe

NP-Difícil e é considerado de difícil resolução de maneira exata, foram desenvolvidos dois algoritmos híbridos, VNS-IG e TS-IG, a fim de obter soluções de forma aproximada de alta qualidade em baixo tempo computacional. Esses algoritmos combinam as meta-heurísticas *Variable Neighborhood Search* (VNS) e Busca Tabu (TB, do inglês *Tabu Search*) com o *Iterated Greedy* (IG). Experimentos computacionais e análises estatísticas foram realizados a fim de comparar o desempenho das versões do PS e dos algoritmos propostos. De acordo com os experimentos computacionais, as versões do PS obtiveram melhorias na qualidade da solução obtida e redução no tempo computacional se comparado a resolução do modelo matemático pelo solver IBM ILOG CPLEX. Além disso os experimentos realizados mostram que algoritmos propostos são significativamente superiores ao melhor algoritmo da literatura (*Scatter Search*) em relação a dois fatores: qualidade das soluções e tempo de execução.

Abstract

ARAUJO, Matheus de Freitas, M.Sc., Universidade Federal de Viçosa, March, 2017. **Efficient methods for non-permutation flow shop scheduling problem with heterogeneous workers**. Adviser: José Elias Claudio Arroyo.

The current work addresses the non-permutation flow shop scheduling problem with heterogeneous workers (Het-FSSP), which is defined as a multicomponent problem, since it combines two problems where the result of one affects the other. Het-FSSP consists of two classical combinatorial optimization problems: machine worker allocation and non-permutation flow shop scheduling (NPFSS). The problem is to allocate heterogeneous workers in machines arranged in series, in which the heterogeneity is due to the time spent by the worker when operating a machine. The allocation of workers defines the periods of execution of the jobs of the NPFSS problem. The goal of the Het-FSSP problem is to minimize the maximum job completion time, which is known as makespan. In order to solve this problem, it is initially proposed to apply the Proximity Search (PS) method to try to determine optimal solutions for the problem using mixed integer programing (MIP) model. This method consists of replacing the objective function with a proximity function and adding a cut-off constraint on the model. Then, by iteration, the new model is resolved and the cut restriction is updated. This ensures that PS limits the search space and identifies optimal solutions. Three PS versions denoted by PS_1, PS_2 and PS_{2RINS} have been developed. Since the problem belongs to the NP-Difficult class and is considered to be difficult to solve exactly, two hybrid algorithms, VNS-IG and TS-IG, were developed in order to obtain approximate

solutions of high quality in low computational time. These algorithms combine the meta-heuristics Variable Neighborhood Search (VNS) and Tabu Search (TS) with Iterated Greedy (IG). Computational experiments and statistical analyzes were performed in order to compare the performance of PS versions and the proposed algorithms. The computational experiments results suggest that the PS versions acquired improvements in the quality of the solution obtained and also reduced computational time spent compared to the resolution of the mathematical model by the IBM ILOG CPLEX solver. In addition, the experiments performed have shown that the proposed algorithms are significantly superior to the best algorithm found in the literature (Scatter Search) in relation to two factors: solution quality and execution time.

Capítulo 1

Introdução

Um dos grandes problemas sociais enfrentados pela humanidade está diretamente ligada ao mercado de trabalho. O desemprego é um problema constante em praticamente todos os países, independente da situação econômica. Elevadas taxas de desemprego geram diversos problemas tanto do ponto de vista do desempregado quanto dos pontos de vista social, econômico e político. Exemplos disso são problemas relacionados a saúde física e mental do desempregado, aumento da criminalidade e a elevação dos casos de violência na sociedade (Reinert, 2001).

Diversos fatores podem influenciar no aumento das taxas de desemprego, tais como: fatores climáticos, desindustrialização, desenvolvimento tecnológico, altos custos com encargos trabalhistas, crise econômica, dentre outros. Alinhado a isso, a falta de profissionais qualificados e experientes para assumirem os postos de trabalhos é um problema recorrente enfrentado pelas empresas e industrias (Barros et al., 1997). No Brasil a taxa de desemprego em 2016 foi de aproximadamente 11,5% segundo os dados do Instituto Brasileiro de Geografia e Estatística (IBGE, 2016) , ficando à frente de países como Peru (6,2%), Argentina (8,5%), Bolívia (7,4%), Estados Unidos (4,8%), China (4,02%) e Japão (3,10%) como mostra a plataforma Trading Economics (IECONOMICS, 2017).

Um ponto relevante a ser tratado, diz respeito a taxa de desemprego sofridas por pessoas com deficiência. Historicamente, tais indivíduos foram excluídos da sociedade, um exemplo disso é a Constituição do Brasil Imperial (Constituição Política do Império do Brasil) de 1824 que retirava a garantia dos direitos previstos

por lei dos deficientes, uma vez que, considerava-os incapazes de realizar qualquer atividade (Paim, 2015). Mesmo após a aprovação da Lei Nº 13.146 (Brasil, 2015), que define o estatuto de pessoas com deficiência, os deficientes ainda enfrentam diversas dificuldades impostas indiretamente pela sociedade, como a falta de acessibilidade, a dificuldade de qualificação e taxas de desemprego superior as enfrentadas pelo restante da população.

Segundo os dados do último censo demográfico do IBGE (2010), o Brasil conta com aproximadamente 45 milhões de brasileiros que sofrem com alguma deficiência (visual, auditiva, motora, mental ou intelectual), mundialmente são aproximadamente 700 milhões de deficientes segundo a Organização Internacional do Trabalho, sendo cerca 500 milhões em idade laboral. Ainda segundo o censo demográfico, 53% dos deficientes brasileiros em idade ativa (maiores de 10 anos) estão desocupadas, ou seja, estão sem estudar ou trabalhar. Esses números, juntamente com a descriminação, explicam a dificuldade dos deficientes de ingressarem no mercado de trabalho.

Diversos autores como Antunes (2004), Giordano (2000) e Costa (2001) apontam o trabalho como sendo fundamental para a existência social das pessoas, além disso, o trabalho aumenta a autoestima, traz efeitos psicológicos positivos e traz aos deficientes maior segurança ao enfrentar as barreiras sociais. É preciso reconhecer a deficiência enquanto uma das muitas formas corporais de estar no mundo e que estas pessoas necessitam de condições favoráveis para levar adiante a vida. A deficiência não significa isolamento ou sofrimento, não há uma sentença biológica que determina o fracasso do deficiente, o que existem são contextos pouco sensíveis à compreensão da diversidade corporal como diferente estilo de vida (Diniz, 2007).

Tendo em vista esse cenário, diversos países têm implantando políticas de inclusão social dos deficientes, como é o caso da lei brasileira N. 7.853 que assegura os direitos básicos dos cidadãos brasileiros, dentre eles, o acesso à educação e ao trabalho.

> "A empresa com cem ou mais empregados está obrigada a preencher de dois a cinco por cento de seus cargos com beneficiários da Previdência

> Social reabilitados ou com pessoa portadora de deficiência habilitada", Brasil (1999).

Países como Espanha, Japão e Inglaterra possuem Centros de Trabalho Protegido para Deficientes (CTDs), que se trata de empresas sem fins lucrativos que buscam empregar o maior número possível de deficientes com o objetivo de qualificar e treinar essas pessoas, para que, posteriormente sejam absorvidas por empresas e industrias convencionais (Chaves, 2009). No Brasil, existem programas como os mantidos pela Associação de Assistência à Criança Deficiente (AACD, 2001) e o Centro de Vida Independente do Rio de Janeiro (CVI-Rio, 1982) que oferecem educação e capacitação profissional a cidadãos com deficiência.

A partir da concepção de que os deficientes precisam ser eficientes e competitivos, não só pela sua sobrevivência no mundo, mas também para serem capazes de crescer no mercado de trabalho. Esta pesquisa busca contribuir com as empresas que se utilizam da mão de obra dos portadores de necessidades especiais (como as CTDs) e o sistemas de produção *flow shop*. Melhorando assim a produtividade das empresas que utilizam-se destes sistemas, tornando-as competitivas por meio da alocação eficiente das tarefas, e as condições de trabalho dos deficientes, que seriam alocados em postos nos quais suas diferenças e/ou falta de experiência não influenciariam na qualidade de sua mão de obra.

O problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH), proposto por Benavides et al. (2014), é considerado como um problema multicomponente, problemas dessa natureza combinam dois ou mais problemas individuais de otimização, na qual, essa combinação torna-os capazes de influenciar nos resultados de uns dos outros. Nesse caso, a combinações das melhores soluções dos problemas individual, não representa, necessariamente, a melhor solução para o problema completo (Bonyadi et al., 2013). O problema de FSNPTH consistem na união de um problema de transporte (Hillier & Lieberman, 2013) com um problema da teoria de *scheduling* (Brucker, 2007).

O primeiro problema consiste em designar trabalhadores heterogêneos às máquinas, nas quais a heterogeneidade dos trabalhadores é definida em relação ao tempo que cada trabalhador gasta para operar uma determinada máquina. O segundo trata-se do problema de *flow shop scheduling* não permutacional (FSNP),

que busca sequenciar de maneira eficiente um conjunto de tarefas não preemptivas em um conjunto de máquinas dispostas em série, visando reduzir o tempo máximo de conclusão das tarefas, conhecido como *makespan*.

Sendo assim, o problema de FSNPTH pode ser definido em alocar um conjunto de trabalhadores $W = \{1, 2, ..., w\}$ em um conjunto $M = \{1, 2, ..., m\}$ de máquinas, sendo o número de trabalhadores (w) igual ao número de máquina (m), onde, cada máquina deve ser operada por um único trabalhador. Ao definir a alocação de trabalhadores às máquinas, define-se também, os tempos de processamento das tarefas do sistema de produção *flow shop*, isso comprova a interdependência entre os problemas, uma das características dos problemas multicomponentes (Bonyadi et al., 2013). Partindo disso, tem-se o problema de FSNP, onde, existe um conjunto $N = \{1, 2, ..., n\}$ de tarefas que devem ser processadas no conjunto máquinas dispostas em séries. Cada tarefa j é composta por um conjunto de m operações consecutivas $O_{1j}, O_{2j}, ..., O_{mj}$, a operação O_{ij} representa o tempo da operação da tarefa j na máquina i. O objetivo do problema FSNPTH é definir a alocação dos trabalhadores às máquinas de maneira que o sequenciamento das tarefas nas máquinas tenha o menor tempo máximo de conclusão das tarefas (*makespan*).

Neste trabalho é aplicado o método *Proximity Search* (PS), proposto por Fischetti & Monaci (2014) para resolver problemas de programação linear inteira mista 0-1 (PLIM). O PS é considerado um método exato para resolução dos problemas nos quais é aplicado, uma vez que, consegue provar a otimalidade das soluções encontradas. Esse método foi aplicado no problema de FSNPTH com objetivo de encontrar soluções ótimas, uma vez que, a resolução do modelo matemático resolvido pelo solver IBM ILOG CPLEX (IBM, 2011) se mostrou ineficiente para as instâncias propostas.

Por se tratar de um problema de otimização combinatória de difícil resolução de maneira exata (Benavides et al., 2014), a utilização de algoritmos baseados em meta-heurísticas é, geralmente, uma boa opção para encontrar soluções de boa qualidade em tempo aceitável. Neste trabalho são desenvolvidos dois algoritmos híbridos: o VNS-IG, baseado nas meta-heurísticas *Variable Neighborhood Search* (VNS) (Mladenović & Hansen, 1997) e *Iterated Greedy* (IG) proposto por Benavides & Ritt (2016); e o TS-IG, baseado nas meta-heurísticas Busca Tabu

(TS, do inglês *Tabu Search*) (Glover, 1989) e IG. Os métodos e algoritmos propostos são analisados e comparados estatisticamente sobre um conjunto de instâncias geradas para o problema a fim de provar seu bom desempenho em relação aos algoritmos e métodos disponíveis na literatura.

1.1 Objetivos

O objetivo deste trabalho é aplicar técnicas de resolução exatas e desenvolver algoritmos baseados em meta-heurísticas que sejam eficientes na resolução do problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos, produzindo soluções de alta qualidade em tempo computacional aceitável e que possam ser comparados com os métodos disponíveis na literatura.

1.1.1 Objetivos Específicos

Para alcançar o objetivo geral é necessário que sejam atingidos os objetivos específicos a seguir:

1. Realizar revisões bibliográficas que abordam problemas relacionados, obtendo assim os conceitos das principais estratégias de resolução aplicadas aos problemas relacionados;
2. Implementar o modelo matemático proposto na literatura para resolução do problema de maneira exata;
3. Implementar os métodos propostos para resolução do problema de maneira exata;
4. Propor e implementar novos algoritmos baseados em abordagens heurísticas para este problema;
5. Realizar experimentos computacionais para calibração dos parâmetros utilizados nas estratégias implementadas;
6. Realizar experimentos computacionais utilizando instâncias propostas pela literatura para o problema a fim de comprovar e comparar a eficiência dos algoritmos desenvolvidos;

7. Analisar os resultados obtidos e validar o desempenho dos métodos através de testes estatísticos;

8. Realizar uma análise comparativa entre os resultados obtidos a partir dos métodos e algoritmos propostos e os resultados da abordagem heurística da literatura.

1.2 Estrutura do trabalho

Esse primeiro capítulo apresentou uma introdução geral do problema a ser tratado nesta dissertação. Nos próximos capítulos serão apresentados dois artigos científicos. O primeiro artigo apresenta a utilização do método *Proximity Search* para resolução do problema de maneira exata. O segundo apresenta dois algoritmos híbridos, VNS-IG e TS-IG, propostos para resolução do problema de maneira aproximada e em tempo computacional aceitável.

São eles:

(a) Capítulo 2: Método *Proximity Search* para a resolução do problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH). Artigo científico publicado nos Anais do XLVIII SBPO – Simpósio Brasileiro de Pesquisa Operacional.

(b) Capítulo 3: Dois algoritmos híbridos para o problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH). Artigo científico a ser submetido para a revista *Computers & Industrial Engineering* - Elsevier;

No quarto e último capítulo é apresentado uma conclusão geral.

Capítulo 2

Método *Proximity Search* para a resolução do problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH).

Artigo científico publicado nos Anais do XLVIII SBPO – Simpósio Brasileiro de Pesquisa Operacional.

XLVIII SBPO

Anais do XLVIII SBPO
Simpósio Brasileiro de Pesquisa Operacional
Vitória, ES, 27 a 30 de setembro de 2016.

Método *Proximity Search* para a resolução do problema de flow shop scheduling não permutacional com trabalhadores heterogêneos (FSNPTH).

Matheus de Freitas Araujo
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brasil
matheus.f.araujo@ufv.br

José Elias Claudio Arroyo
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brasil
jarroyo@dpi.ufv.br

André Gustavo dos Santos
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brasil
andre@dpi.ufv.br

RESUMO

Esse trabalho aplicar o método *Proximity Search* (PS) para a resolução do problema FSNPTH. Este problema pode ser classificado como um problema multicomponente, uma vez que trata de dois problemas distintos na qual a resolução de um implica diretamente no resultado do outro. O problema consiste em: alocar trabalhadores heterogêneos em máquinas, nas quais a heterogeneidade dos trabalhadores é definida em relação ao tempo gasto para operar uma determinada máquina; sequenciar tarefas em máquinas em séries de forma que minimize o tempo de conclusão da última tarefa no processo (makespan). PS é um método genérico que faz uso do modelo de Programação Inteira do problema para melhorar iterativamente a solução. O PS modifica a função objetivo para reduzir o espaço de busca. Neste trabalho, o desempenho do PS é comparado com o solver CPLEX. Os resultados mostraram que o PS determina boas soluções em curto espaço de tempo.

PALAVRAS CHAVE. Flow Shop Scheduling, *Proximity Search*, Programação Matemática.

IND - PO na Indústria, OC - Otimização Combinatória, PM - Programação Matemática

ABSTRACT

This work aims to apply the method Proximity Search (PS) to solve the problem FSNPTH. This problem can be classified as a multicomponent problem, since these are two separate problems, which the resolution of a directly gives the results of another. The problem consists in: allocating heterogeneous workers in machines in which the heterogeneity of workers is defined in relation to the time spent to operate a particular machine; sequencing jobs on machines in series that minimizes the time to complete the last task in the process (makespan). PS is a generic method that makes use of the problem of Integer Programming model to iteratively improve a reference solution. The PS changes the objective function to reduce the search space. In this study, the performance of the PS is compared with the CPLEX solver solving the original problem model. The results showed that the PS provides good solutions in a short time.

KEYWORDS. Flow Shop Scheduling, Proximity Search, Mathematical Programming.

IND - PO Industry, CO - Combinatorial Optimization, PM - Mathematical Programming

XLVIII SBPO
Anais do XLVIII SBPO
Simpósio Brasileiro de Pesquisa Operacional
Vitória, ES, 27 a 30 de setembro de 2016.

1. Introdução

Os problemas de flow shop scheduling ganharam espaço não só na academia, mas também no mercado de trabalho, uma vez que esses problemas refletem diretamente no sistema de produção de diversas empresas. Entretanto, apenas os problemas de flow shop scheduling não têm atendido completamente as empresas, uma vez que este problema busca apenas alocar de maneira eficiente um conjunto de tarefas não preemptivas em um conjunto de máquinas dispostas em série, visando minimizar algumas medidas de desempenho, tal como o tempo máximo de conclusão (Makespan). Sendo assim tem-se tornado cada vez mais comum a combinação de dois ou mais problemas, para suprir melhor a necessidade de resolução dos problemas enfrentados no cotidiano das empresas. Um exemplo disso é a abordagem apresentada por [Benavides et al. 2014] que combina o problema de flow shop não permutacional com a designação de trabalhadores com diferentes habilidades em postos de trabalhos ou máquinas.

O problema de flow shop não permutacional com trabalhadores heterogêneos (FSNPTH) surgiu a partir da necessidade de alocação de trabalhadores deficientes com diferentes habilidades em postos de trabalhos, de forma que a heterogeneidade dos trabalhadores interfira o mínimo possível no resultado final do sistema de produção.

Segundo o Instituto Brasileiro de Geografia e Estatística – IBGE, 23% da população brasileira sofre de alguma deficiência (visual, auditiva, motora, mental ou intelectual) [Brasil 2010]. O IBGE ainda afirma que existem cerca de 23,7 milhões pessoas com deficiência em idade ativa desempregadas, mesmo com as políticas para integração de pessoas portadoras de deficiência, como a lei 7.853 que assegura o acesso ao trabalho. Esses números comprovam a relevância desse trabalho, visto que os deficientes precisam ser eficientes e competitivos ao exercer suas atividades, não só pela sua sobrevivência no mundo, mas também para serem capazes de crescer no mercado de trabalho. Com isso visa-se melhorar a inclusão social dos deficientes, designando-os em postos de trabalhos nos quais suas deficiências interfiram o minimo na atividade.

Na literatura, diversos pesquisadores abordam o problema de flow shop scheduling tais como: Ruiz e Maroto [Ruiz e Maroto 2005] e Potts e Trusevich [Potts e Strusevich 2009]. Muitos autores apresentam metaheuristicas para resolução do problema de maneira aproximada, como: Simulated Annealing [Tandon et al. 1991], Iterated Greedy [Ying 2008], Busca Tabu [Brucker et al. 2003, Liao et al. 2006], Colônia de Formigas [Yagmahan e Yenisey 2010, Rossi e Lanzetta 2013] e algoritmos híbridos que combinam diferentes meta-heuristicas [Lin e Ying 2009].

Esse trabalho apresenta e comprova a eficiência do método *Proximity Search* proposto por [Fischetti e Monaci 2014] para a resolução do problema FSNPTH.

2. Definição do Problema

O problema FSNPTH definido por [Benavides et al. 2014] consiste em alocar trabalhadores heterogêneos à máquinas, nas quais a heterogeneidade dos trabalhadores é definida em relação ao tempo que cada um gasta para operar uma determinada máquina, e a sequenciar tarefas nas máquinas em séries de forma que o makespan seja minimizado.

Sendo assim, existe um conjunto de trabalhadores $W = \{1, 2, ..., w\}$ que deve ser alocado em um conjunto $M = \{1, 2, ..., m\}$ de máquinas, sendo $w = m$, ou seja, cada trabalhador deve ser alocado em uma única máquina. Definida a alocação de trabalhadores às máquinas, tem-se o problema de flow shop scheduling não permutacional, onde existe um conjunto $N = \{1, 2, ..., n\}$ de tarefas na qual que devem ser processadas no conjunto máquinas dispostas em séries. Cada tarefa j é composto por um conjunto de m operações consecutivas $O_{1j}, O_{2j}, ..., O_{mj}$, onde O_{ij} é a operação da tarefa j na máquina i. O tempo de processamento de cada operação das tarefas depende diretamente do trabalhador alocado à máquina, ou seja, se o trabalhador k é alocado à máquina i, o tempo da operação O_{ij} é t_{ijk}. Com isso busca-se encontrar um conjunto π com m sequências (uma para cada máquina) que minimize o makespan. O problema FSNPTH é NP-Difícil, já que o problema de flow shop scheduling permutacional com $m \geq 3$ é NP-Difícil [Garey et al. 1976]. Na Tabela 1 é apresentada uma instância para o problema FSNPTH com 4 tarefas, 4 máquinas e 4

trabalhadores, no qual os valores representam o tempo de processamento de cada tarefa j em cada máquina i dado um trabalhador k (t_{ijk}). Os valores de $t_{ijk} = inf$ indicam que o trabalhador k não pode ser alocado (ou operar) à máquina i.

Tabela 1: Instância do problema de flow shop scheduling não permutacional com trabalhadores heterogêneos.

Tarefas	m_1				m_2				m_3				m_4			
	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4
j_1	1	2	3	1	2	4	2	inf	2	4	2	2	2	2	2	2
j_2	1	1	4	3	1	1	4	inf	2	2	2	4	2	1	3	4
j_3	2	1	2	3	2	2	1	inf	2	1	2	4	4	2	2	3
j_4	2	5	2	3	1	4	2	inf	1	4	2	2	2	3	2	1

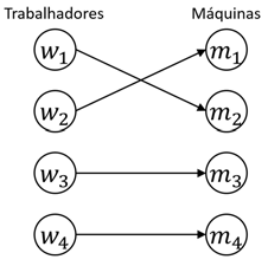


Figura 1: Alocação dos trabalhadores para a instância acima.

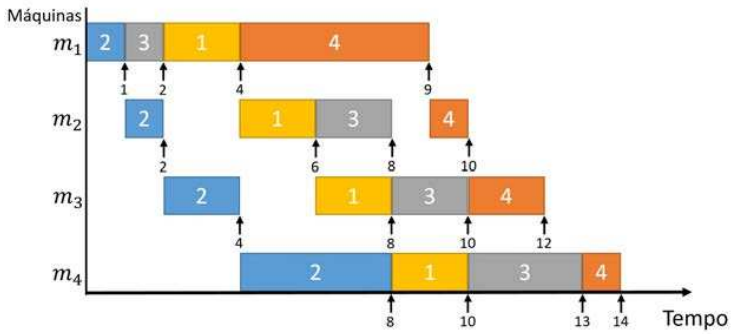


Figura 2: Alocação ótima das tarefas para a instância acima.

Na Figura 1 é apresentada a designação dos trabalhadores às máquinas e na Figura 2 é apresentado o sequenciamento ótimo das tarefas nas máquinas da solução ótima para a instância em questão. A Figura 2 apresenta o conjunto π de m sequências do problema FSNPTH com o valor de makespan igual a 14 ($C_{max} = 14$). A sequência $\pi_1 = \{2, 3, 1, 4\}$ é executada na máquina 1, $\pi_2 = \{2, 1, 3, 4\}$ é executada na máquina 2, as sequências $\pi_3 = \{2, 1, 3, 4\}$ e $\pi_4 = \{2, 1, 3, 4\}$ são executadas nas máquinas 3 e 4 respectivamente. Observa-se que mesmo a designação não sendo ótima, levando em consideração a média ou a soma dos tempos de processamentos das tarefas, o valor makespan (tempo de conclusão da última tarefa no processo) é mínimo para este sequenciamento. Isso acontece por se tratar de um problema multicomponente, ou seja, um problema que consiste em dois ou mais problemas individuais [Bonyadi et al. 2013]. Em problemas dessa natureza duas características podem ser observadas: a combinação e a interdependência. A combinação

é a característica que determina que um problema seja a combinação de mais de um problema, já a interdependência faz com que os problemas interfiram no resultado uns dos outros.

3. Modelagem Matemática de Programação Inteira Mista

Em [Benavides et al. 2014] é proposta uma modelagem matemática para o problema FSNPTH na qual existem m máquinas, w trabalhadores, (onde $m = w$) e n tarefas. O modelo apresentado a seguir utiliza o seguinte parâmetro:

- MG é um valor suficientemente grande definido por $\sum_{i=1}^{m}\sum_{j=1}^{n} max(t_{ijk} \forall k \in W)$;

O modelo utiliza as seguintes variáveis de decisão:

- C_{max}: Tempo máximo de conclusão das tarefas ou makespan;
- x_{ij}: Tempo de início da tarefa j na máquina i;
- p_{ij}: Tempo de processamento da tarefa j na máquina i;
- z_{ik}: Define a alocação do trabalhador k na máquina i, sendo $z_{ik} = 1$ se o trabalhador k é alocado a máquina i e $z_{ik} = 0$ caso contrário;
- $y_{ijj'}$: Define a precedência das tarefas, sendo $y_{ijj'} = 1$ se a tarefa j precede a tarefa j' na máquina i e $y_{ijj'} = 0$ caso contrário;

O modelo é apresentado a seguir:

$$\min C_{max} \tag{1}$$

sujeito a:

$$x_{Mj} + p_{Mj} \leq C_{max}, \qquad \forall j \in N \tag{2}$$

$$x_{ij} + p_{ij} \leq x_{i+1j}, \qquad \forall i \in M-1, \forall j \in N \tag{3}$$

$$x_{ij} + p_{ij} \leq x_{ij'} + MG(1 - y_{ijj'}), \qquad \forall i \in M, \forall j \neq j' \in N \tag{4}$$

$$y_{ijj'} + y_{ij'j} = 1, \qquad \forall i \in M, \forall j \neq j' \in N \tag{5}$$

$$p_{ij} = \sum_{k=1}^{w} t_{ijk} z_{ik} \qquad \forall i \in M, \forall j \in N \tag{6}$$

$$\sum_{k=1}^{w} z_{ik} = 1, \qquad \forall i \in M \tag{7}$$

$$\sum_{i=1}^{m} z_{iw} = 1, \qquad \forall w \in W \tag{8}$$

$$x_{ij} \geq 0, \qquad \forall i \in M, \forall j \in N \tag{9}$$

$$y_{ijj'} \in \{0,1\}, \qquad \forall i \in M, \forall j \neq j' \in N \tag{10}$$

$$z_{ik} \in \{0,1\}, \qquad \forall i \in M, \forall k \in W \tag{11}$$

A função objetivo denifida em (1) é minimizar o valor do makespan (C_{max}). A restrição (2) determina o C_{max}. O conjunto de restrições (3) e (4) dizem respeito ao tempo de início de cada tarefa em cada máquina de acordo com a precedência de tarefa. A restrição de precedência das tarefas é definida pela Equação (5). A restrição (6) define o valor de p_{ij} a partir da alocação dos trabalhadores. O conjunto de restrições (7) e (8) garante a designação dos trabalhadores aos postos de trabalhos ou máquinas. A restrição (9) define a não negatividade das variáveis x_{ij}. O conjunto de restrições (10) e (11) define $y_{ijj'}$ e z_{ik} como variáveis binárias.

XLVIII SBPO
Anais do XLVIII SBPO
Simpósio Brasileiro de Pesquisa Operacional
Vitória, ES, 27 a 30 de setembro de 2016.

4. Proximity Search

O *Proximity Search* (PS) é um método proposto por [Fischetti e Monaci 2014] para resolver problemas de programação linear inteira mista 0-1 (PLIM). Este método utiliza uma estratégia de resolução cujo objetivo é, a partir de uma solução inicial viável, encontrar uma sequência de soluções melhoradas para um dado modelo de PLIM do problema. O funcionamento do PS consiste em substituir a função objetivo do modelo de programação inteira mista por uma função de proximidade e utilizar a resolução deste novo modelo como heurística de refinamento. A proposta do PS é suprir a deficiência dos métodos enumerativos, geralmente utilizados para resolução de problemas de PLIM, para quais não consegue encontrar soluções de boa qualidade em um curto espaço de tempo. Sendo assim, o PS tem se mostrado mais eficiente para resolver problemas de grande porte do que os métodos enumerativos como o branch-and-bound.

O PS inicia a busca a partir de uma solução inicial x', geralmente encontrada por uma heurística, após é adicionada uma restrição de corte (definida pela Equação 12) ao modelo de PLIM. Sendo x e x' duas soluções para o problema, $f(x)$ e $f(x')$ representam os valores do makespan (funções objetivos) e $\theta \geq 0$ é uma constante denominada parâmetro de corte.

$$f(x) \quad \geq \quad f(x') - \theta, \tag{12}$$

Essa restrição garante que apenas soluções melhores que a solução corrente possam ser encontradas quando o modelo é resolvido, com isso o espaço de busca é reduzido consideravelmente.

O próximo passo do PS é substituir a função objetivo do modelo por uma função de proximidade. Os autores [Fischetti e Monaci 2014] propõem a utilização da distância de Hamming como função de proximidade. A ideia por trás da função de proximidade é guiar o algoritmo pelo espaço busca. Com isso aumenta-se a chance de encontrar soluções melhores que a solução inicial. A distância de Hamming é definida pela Equação (13) e busca determinar a distância entre as soluções x e x' formadas por um conjunto de variáveis binárias. Nesse caso, as variáveis binárias $y_{ijj'}$ e z_{ik}, definidas no modelo apresentado na seção 3 são utilizadas para definir a distância entre as soluções x e x'. A distância de Hamming na prática determina o número de variáveis binárias cujo valores sejam distintos entre as soluções x e x'. Em um problema com 2 trabalhadores e 2 máquinas por exemplo, se a solução x tem o trabalhor 1 alocado na máquina 2 ($z_{21} = 1$ e $z_{11} = 0$) e na solução x' o trabalhador 1 é alocado na máquina 1 ($z_{11} = 1$ e $z_{21} = 0$), então o valor de $\Delta(x, x')$ é incrementado em 2 uma vez que as variáveis z_{11} e z_{21} possuem valores distintos nas soluções x e x'. Se as soluções x e x' forem iguais $\Delta(x, x') = 0$.

$$\Delta(x, x') \quad = \quad \sum_{j \in J : x^*_j = 0} x_j \quad + \sum_{j \in J : x^*_j = 1} (1 - x_j) \tag{13}$$

Algorithm 1 Proximity Search.

1: $x' \leftarrow$ Solução Inicial
2: **enquanto** critério de parada não seja atingido **faça**
3: Adiciona a restrição de corte $f(x) \geq f(x') - \theta$ ao modelo
4: Substituir a função objetivo $f(x)$ do modelo pela função de proximidade $\Delta(x, x')$
5: $x^* \leftarrow$ executa o solver CPLEX até que a condição de parada seja atingida.
6: **se** x^* é uma solução viavel **então**
7: $x^* \leftarrow argmin\{f(x) : g(x) \geq 0, x_j = x^*_j, \forall j \in J\}$
8: **fim se**
9: Atualizar $\Delta(x, x')$ fazendo $x' \leftarrow x^*$ e/ou atualiza θ.
10: **fim enquanto**

XLVIII SBPO
Anais do XLVIII SBPO
Simpósio Brasileiro de Pesquisa Operacional
Vitória, ES, 27 a 30 de setembro de 2016.

O pseudocódigo do *Proximity Search* é descrito pelo Algoritmo 1. O PS necessita de uma solução inicial (Linha 1), então na Linha 3 é adicionada a restrição de corte baseada no valor da solução corrente (x') e o valor de θ. Na Linha 4 a função objetivo do modelo de programação inteira mista é substituída pela função de proximidade $\Delta(x, x')$, que é definida de acordo com a distância de Hamming. Em seguida, na Linha 5 o solver IBM ILOG CPLEX é utilizado para resolver o novo modelo. Na Linha 6 é verificado se o solver Cplex encontrou alguma solução para o novo modelo, caso encontre, na Linha 7 é calculado o valor da função objetivo $f(x^*)$ do modelo original. Em seguida é necessário atualizar a função de proximidade $\Delta(x, x')$ bem como a solução corrente, que deve ser substituída pela solução encontrada pelo modelo (Linha 9).

A partir do algoritmo descrito anteriormente, foram implementadas três versões do PS, PS_1, PS_2 e PS_{2RINS} [Fischetti e Monaci 2014]. O PS_1 é o Algoritmo 1, entretanto nas versões PS_2 e PS_{2RINS} a função de corte apresentada pela Equação (12) é substituída pela função de corte definida na Equação (14), onde z é uma variável de folga contínua ($z >= 0$) e a função de proximidade é substituída pela função de proximidade definida na Equação (15) onde M é um valor positivo suficientemente grande se comparado com valor obtido por Δ (nesse caso é usado $M = (j^2 * m + m^2 + 1)$, que é o número de variáveis binárias utilizadas no modelo mais um). Com a variável de folga z aplicada na função de proximidade e na restrição de corte, o valor da solução incumbente do modelo é rapidamente definida e pode ser igual à solução corrente x'. Essa situação seria inválida, pois espera-se encontrar uma solução x melhor e diferente de x'. Para isso, a constante M vai penalizar a solução com uma função de proximidade muito alta. Ter uma solução incumbente logo no início da resolução do modelo matemático pode melhorar o desempenho e permitir a utilização de heurísticas de refinamento implementadas internamente pelo CPLEX.

$$f(x) \quad \geq \quad f(x') - \theta + z \tag{14}$$

$$\Delta(x, x') + Mz \tag{15}$$

A versão PS_{2RINS} é diferente das outras duas versões, uma vez que utiliza a heurística de refinamento RINS(Relaxation Induced Neighborhood Search), introduzida por [Danna et al. 2005]. A biblioteca ILOG Concert (utilizada para nas implementações do PS) por padrão já implementa essa heurística, sendo assim, nessa versão do PS, o RINS é apenas habilitada durante a execução.

5. Experimentos computacionais

Os experimentos computacionais foram realizados a fim de demonstrar o desempenho do PS em relação a resolução do modelo sem essa técnica. Nos testes foram executadas as 3 versões do PS (PS_1, PS_2 e PS_{2RINS}) e o MODELO. O MODELO refere-se à implementação e resolução do modelo matemático de programação inteira mista definido anteriormente. Esse método então é utilizado posteriormente para realizar as comparações. As implementações do MODELO e do PS foram desenvolvidas utilizando a biblioteca ILOG Concert, que permite implementar modelos matemáticos na linguagem C++ e então resolver pelo solver IBM ILOG CPLEX.

Os experimentos computacionais foram realizados em um Cluster onde cada nó contém 2 processadores Intel(R) Xeon(R) CPU X5650 (12M Cache, 2.66 GHz, 6.40 GT/s Intel(R) QPI, 6 cores, 12 threads) e 24 GB de RAM DDR3 1333 MHz. Entretanto os experimentos foram executados utilizando apenas 1 processador, 4 threads e 8GB de memória RAM. Em todos os experimentos foram fixados o tempo de execução em 3600 segundos para cada instância e o valor de θ foi fixado em 1 (garantindo que o PS prove a otimalidade da solução).

Para cada uma das versões do PS, foram executados testes variando os pesos das variáveis que representam os trabalhadores (z_{ik}) na função de proximidade. Foram testados 4 pesos (1, 10, 50, 100). Por se tratar de um problema multicomponente, ao atribuir pesos nas variáveis dos trabalhadores, a distância entre duas soluções (x e x') aumenta quando um trabalhador é trocado de máquina, uma vez que ao trocar dois trabalhadores de máquina o tempo das operações O_{ij} são

Tabela 2: Resultados (média do makespan) obtidos para cada implementação do PS.

Peso	Proximity Search		
	PS_1	PS_2	PS_{2RINS}
1	9802,4	**9468,4**	9305,8
10	9756,9	9502,5	**9274,3**
50	**9752,0**	9492,3	9276,0
100	9809,2	9566,7	9311,3

alterados. Com os pesos na função de proximidade o PS é forçado a permutar com mais frequência as tarefas ao invés de permutar os trabalhadores.

5.1. Resultados Computacionais

Para avaliar os resultados encontrados pelas implementações do PS e o MODELO, o valor de $RDP(\%)$ (Desvio Percentual Relativo) é calculado através da Equação (16), onde $Makespan$ é o valor do makespan econtrado pelo método em questão e ub é o melhor valor conhecido para determinada instância. As instâncias propostas por [Benavides et al. 2014] são baseadas nas instâncias Het-FSSP (Heterogeneous Flow Shop Scheduling Problem) de Carlier [Carlier 1978] e nas instâncias de flow shop scheduling de Taillard [Taillard 1993], sendo assim, os valores do ub para essas instâncias são encontrados em [Taillard 2004] e [Carlier e Pinson 1990].

$$RDP(\%) = \frac{Makespan - ub}{ub} \times 100 \quad (16)$$

Para análise dos resultados encontrados para as instâncias de Taillard, grupos de dez instâncias são formados e nesses casos a média é aparesentada como makespan final.

As Tabelas 3, 4, 6 e 7 apresentam os valores do $RDP(\%)$ para cada instância e cada método. Sendo que nas implementaçãos do PS utilizam-se os pesos que obtiveram menor média do makespan para todas as instâncias (como mostrado na Tabela 5). Os melhores valores encontrados para $RDP(\%)$ estão destacados em negrito.

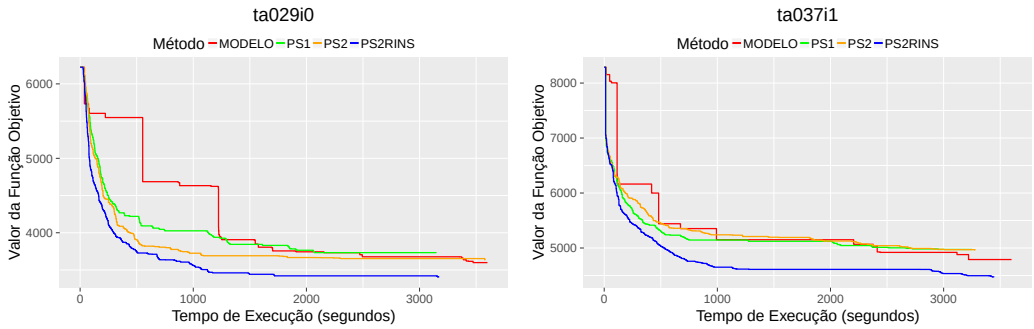
Para as instâncias de Carlier (consideradas pequenas e apresentado na Tabela 3 e Tabela 4) os resultados entre o MODELO e as implementações do PS são semelhantes, entretanto, o MODELO consegue ser melhor que o do PS em 6 instâncias, em contrapartida o PS é superior em 7 instâncias, ou seja, pelo menos uma das versões do PS consegue encontrar soluções melhores que o MODELO. Durante a execução do MODELO foram encontradas 13 soluções ótimas (em negrito), por conseguinte o PS conseguiu encontrar apenas 12 dessas soluções. A Tabela 5 apresenta o tempo gasto por cada método para encontrar a solução ótima, nesses casos pelo menos uma implementação do PS se mostra mais eficientes em relação ao tempo para encontrar a solução ótima.

Para as instâncias de Taillard (consideradas grandes e tem os resultados apresentados na Tabela 6 e Tabela 7) o PS_{2RINS} se sobressai melhor em 22 grupos, o PS_2 é melhor em 3 grupos e o MODELO tem vantagem sobre 11 grupos, no total de 36 grupos. Entretanto nos 3 grupos em que o PS_2 é melhor, o PS_{2RINS} é o método que possui o segundo melhor $RDP(\%)$. Para estes grupos pode-se afirmar que independente das duas implementações do PS que utilizar, o resultado obtido é melhor que o encontrado pela resolução do modelo matemático utilizando o software IBM ILOG CPLEX. Além disso, o MODELO após 3600 segundos não consegue encontrar solução viável para 18 instâncias, contudo todas as implementações do PS conseguiram encontrar soluções viáveis para todas as instâncias rapidamente (nos piores casos o tempo médio gasto para o PS encontrar uma solução viável é de 600 segundos).

Como já citado anteriormente a principal característica do PS é encontrar uma quantidade de soluções melhoradas logo nos primeiros instantes de busca. Para comprovar essa característica, foram selecionadas 2 instâncias do conjunto de Taillard. A cada melhoria obtida pelo PS e MODELO foram armazenados os tempos gastos para obter a solução. Ao final foi gerado um gráfico $tempo(s)$ X $Makespan$ para cada instância. Os resultados são apresentados na Figura 3.

Tabela 3: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Carlier com tempo de processamento entre t e $2t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
car1i0.txt	11x5	9952	**0,00**	**0,00**	**0,00**	0,26
car1i1.txt	11x5	9952	**0,00**	**0,00**	**0,00**	0,14
car1i2.txt	11x5	9952	**0,00**	**0,00**	**0,00**	**0,00**
car2i0.txt	13x4	10224	**0,00**	**0,00**	**0,00**	**0,00**
car2i1.txt	13x4	10224	**0,00**	**0,00**	**0,00**	**0,00**
car2i2.txt	13x4	10398	**0,00**	**0,00**	**0,00**	**0,00**
car3i0.txt	12x5	10268	1,26	**1,19**	1,26	2,44
car3i1.txt	12x5	10359	0,37	1,54	**0,00**	0,37
car3i2.txt	12x5	10359	0,37	**0,00**	1,54	**0,00**
car4i0.txt	14x4	11613	**0,00**	**0,00**	2,63	**0,00**
car4i1.txt	14x4	11846	**0,00**	**0,00**	0,70	**0,00**
car4i2.txt	14x4	11876	**0,13**	0,99	1,91	1,91
car5i0.txt	10x6	10589	**0,00**	1,84	0,34	0,34
car5i1.txt	10x6	10589	**0,00**	**0,00**	4,04	**0,00**
car5i2.txt	10x6	10848	**0,00**	**0,00**	**0,00**	**0,00**
car6i0.txt	8x9	11044	**0,00**	0,24	0,45	**0,00**
car6i1.txt	8x9	11044	**0,00**	**0,00**	**0,00**	**0,00**
car6i2.txt	8x9	11118	**0,00**	1,74	1,56	**0,00**
car7i0.txt	7x7	8558	**0,00**	**0,00**	**0,00**	**0,00**
car7i1.txt	7x7	8558	**0,00**	**0,00**	**0,00**	**0,00**
car7i2.txt	7x7	8558	**0,00**	**0,00**	**0,00**	**0,00**
car8i0.txt	8x8	11533	**0,36**	1,61	**0,36**	1,09
car8i1.txt	8x8	11659	**0,00**	0,17	0,79	0,72
car8i2.txt	8x8	11742	**0,00**	**0,00**	**0,00**	**0,00**
Médias Totais		10535,96	**0,10**	0,39	0,65	0,30



(a) Instância com m = 10 e n = 20

(b) Instância com m = 20 e n = 20

Figura 3: Atualização da solução de diferentes instâncias ao decorrer de 3600 segundos (Tempo(s) X Makespan) usando o $MODELO$, PS_1, PS_2 e PS_{2RINS}.

Tabela 4: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Carlier com tempo de processamento entre t e $5t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
car1I0.txt	11x5	19508	**0,00**	**0,00**	**0,00**	**0,00**
car1I1.txt	11x5	19831	**0,00**	**0,00**	**0,00**	**0,00**
car1I2.txt	11x5	19831	**0,00**	**0,00**	**0,00**	**0,00**
car2I0.txt	13x4	19876	**0,00**	0,08	0,08	0,01
car2I1.txt	13x4	19876	**0,00**	**0,00**	**0,00**	**0,00**
car2I2.txt	13x4	19876	**0,00**	**0,00**	1,37	1,37
car3I0.txt	12x5	19498	0,29	0,29	0,29	**0,00**
car3I1.txt	12x5	19498	**0,29**	1,66	5,08	**0,29**
car3I2.txt	12x5	19498	0,29	**0,00**	0,29	0,29
car4I0.txt	14x4	20381	0,20	**0,00**	0,20	**0,00**
car4I1.txt	14x4	22270	**0,00**	1,03	1,03	1,03
car4I2.txt	14x4	22270	**0,00**	**0,00**	0,05	0,34
car5I0.txt	10x6	18693	**0,00**	1,05	**0,00**	**0,00**
car5I1.txt	10x6	18693	**0,00**	**0,00**	**0,00**	**0,00**
car5I2.txt	10x6	19468	**0,00**	4,56	**0,00**	**0,00**
car6I0.txt	8x9	20070	0,34	6,74	**0,00**	**0,00**
car6I1.txt	8x9	20070	**0,00**	2,57	3,18	6,39
car6I2.txt	8x9	20070	**0,00**	3,13	2,18	6,58
car7I0.txt	7x7	16423	**0,00**	**0,00**	**0,00**	**0,00**
car7I1.txt	7x7	17067	**0,00**	**0,00**	0,32	**0,00**
car7I2.txt	7x7	17257	**0,00**	**0,00**	**0,00**	**0,00**
car8I0.txt	8x8	19159	**0,00**	**0,00**	0,92	**0,00**
car8I1.txt	8x8	19159	**0,00**	1,20	0,92	0,92
car8I2.txt	8x8	19791	**0,00**	2,74	**0,00**	**0,00**
Médias Totais		19505,54	**0,06**	1,04	0,66	0,72

Tabela 5: Tempo gasto pelos métodos $MODELO$, PS_1, PS_2 e PS_{2RINS} para encontrar a solução ótima.

Instância	ub	Tempo em segundos			
		MODELO	PS_1	PS_2	PS_{2RINS}
car1i0.txt	9952	486	41	**14**	-
car1i2.txt	9952	75	67	85	**16**
car6i1.txt	11044	2685	1577	673	**45**
car7i0.txt	8558	30	36	**20**	34
car7i1.txt	8558	13	41	33	**6**
car7i2.txt	8558	12	**4**	9	872
car5I0.txt	18693	168	-	249	**26**
car6I2.txt	20070	1696	-	-	-
car7I0.txt	16423	15	189	646	**2**
car7I1.txt	17067	590	**70**	-	3591
car7I2.txt	17257	46	**2**	20	3
car8I0.txt	19159	500	30	-	**29**
car8I2.txt	19791	1272	-	29	**6**

Tabela 6: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Taillard com tempo de processamento entre t e $2t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
ta001i0–ta010i0	20x5	1741,00	4,95	8,89	7,00	**4,74**
ta001i1–ta010i1	20x5	1765,70	**3,09**	8,42	8,34	6,36
ta001i2–ta010i2	20x5	1769,60	**3,71**	7,08	9,58	6,35
ta011i0–ta020i0	20x10	2103,40	17,44	18,71	14,21	**9,31**
ta011i1–ta020i1	20x10	2178,90	9,71	13,75	10,95	**6,13**
ta011i2–ta020i2	20x10	2180,00	13,43	15,42	9,42	**5,22**
ta021i0–ta030i0	20x20	3037,60	29,69	27,33	21,98	**11,27**
ta021i1–ta030i1	20x20	3141,40	37,69	20,46	15,17	**5,74**
ta021i2–ta030i2	20x20	3124,10	29,21	23,24	19,62	**7,17**
ta031i0–ta040i0	50x5	4007,60	**17,48**	46,80	40,46	34,58
ta031i1–ta040i1	50x5	4034,50	**19,30**	42,77	38,09	33,77
ta031i2–ta040i2	50x5	4037,80	**17,53**	38,02	36,51	27,51
ta041i0–ta050i0	50x10	4358,40	78,06	74,11	72,87	**65,76**
ta041i1–ta050i1	50x10	4499,00	64,31	58,42	60,36	**50,71**
ta041i2–ta050i2	50x10	4500,80	72,63	76,76	77,89	**69,90**
ta051i0–ta060i0	50x20	5565,20	198,35	185,35	176,26	**172,83**
ta051i1–ta060i1	50x20	5344,10	201,31	207,25	202,29	**191,42**
ta051i2–ta060i2	50x20	5335,20	193,91	181,91	176,56	**167,92**
Médias Totais		3484,68	56,21	58,59	55,42	**48,71**

Tabela 7: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Taillard com tempo de processamento entre t e $5t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
ta001I0–ta010I0	20x5	3343,70	**3,78**	13,29	7,79	8,25
ta001I1–ta010I1	20x5	3356,20	**4,24**	11,26	8,37	6,74
ta001I2–ta010I2	20x5	3375,20	**4,81**	9,37	7,31	8,24
ta011I0–ta020I0	20x10	3995,00	18,79	20,35	16,55	**11,32**
ta011I1–ta020I1	20x10	4086,70	16,17	20,21	15,23	**11,60**
ta011I2–ta020I2	20x10	4120,50	16,73	19,82	19,62	**9,74**
ta021I0–ta030I0	20x20	5796,20	49,37	30,06	24,66	**12,56**
ta021I1–ta030I1	20x20	5930,70	37,74	36,15	22,76	**10,40**
ta021I2–ta030I2	20x20	5934,60	30,42	36,02	25,41	**12,89**
ta031I0–ta040I0	50x5	7875,90	**21,75**	34,64	35,68	24,64
ta031I1–ta040I1	50x5	7923,80	**21,31**	37,45	29,61	22,72
ta031I2–ta040I2	50x5	7936,80	**19,85**	34,97	36,25	25,60
ta041I0–ta050I0	50x10	8488,10	87,27	83,20	80,31	**74,85**
ta041I1–ta050I1	50x10	8769,30	88,25	88,27	83,74	**82,29**
ta041I2–ta050I2	50x10	8777,80	87,92	83,49	82,29	**75,77**
ta051I0–ta060I0	50x20	11152,50	204,81	145,93	**125,79**	138,90
ta051I1–ta060I1	50x20	11055,11	220,98	152,95	**137,93**	145,10
ta051I2–ta060I2	50x20	11084,44	283,96	151,99	**136,49**	143,81
Médias Totais		6833,48	67,68	56,08	49,77	**45,86**

Como pode ser observado na Figura 3, as versões do PS conseguem encontrar soluções melhores que o MODELO com menor tempo de execução, e em muitos casos o valor encontrado pelo PS é melhor que o MODELO ao final dos 3600 segundos, sendo o PS_{2RINS} o método que obteve o melhor desempenho. O PS, de modo geral, consegue melhorar a solução, em muitas vezes antes dos 1800 segundos a solução encontrada pelo PS já é melhor que as soluções encontradas pelo MODELO.

6. Conclusão

De acordo com os experimentos realizados na seção anterior, para as instâncias pequenas, como as de Carlier, ambos os métodos utilizados para encontrar as soluções são equivalentes, entretanto o MODELO ainda possui uma pequena vantagem uma vez que conseguiu encontrar um numero maior de soluções ótimas, entretanto o tempo gasto pelo MODELO é muito maior se comparado com as implementações do PS para encontrar essas soluções. Para as instâncias grandes (Taillard) o PS_{2RINS} obteve melhor $RDP(\%)$ para grande parte das instâncias testadas superando as demais implementações do PS e o MODELO, se mostrando a melhor opção para a resolução desse problema.

Entre as versões do PS, de modo geral a versão PS_{2RINS} se sobressai sobre as outras, conseguindo soluções melhores em um menor espaço de tempo, como visto na Figura 3. Em relação ao tempo gasto pelos métodos, as implementações do PS têm vantagem sobre o MODELO, uma vez que conseguem encontrar e/ou melhorar as soluções logo no início da busca. Isso comprova a eficiência do PS sobre o MODELO para resolução do problema de flow shop não permutacional com trabalhadores heterogêneos em um curto espaço de tempo. Além disso, o *Proximity Search* se mostra um método promissor não só para as resoluções de problemas multicomponentes, mas também para a resolução de problemas que possuem um grande espaço de busca.

Como trabalhos futuros, os autores sugerem a elaboração de um sistema para determinar os pesos dinâmicos a serem aplicados as variáveis binárias de decisão utilizados na função proximidade do PS. Além disso avaliar o parâmetro utilizado pela heurística RINS que define o intervalo de nós da árvore de busca onde será aplicada a heurística. Neste trabalho foi adotado que a heurística RINS é aplicada em cada nó da árvore.

7. Agradecimentos

Os autores agradecem à CAPES, CNPq, FAPEMIG e a Divisão de Apoio ao Desenvolvimento Científico e Tecnológico da Universidade Federal de Viçosa.

Referências

Benavides, A. J., Ritt, M. e Miralles, C. (2014). Flow shop scheduling with heterogeneous workers. *European Journal of Operational Research*, 237(2):713–720.

Bonyadi, M. R., Michalewicz, Z. e Barone, L. (2013). The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, p. 1037–1044. IEEE.

Brasil (2010). Secretaria de direitos humanos presidência da república. `http://www.sdh.gov.br/assuntos/pessoa-com-deficiencia/dados-estatisticos/pesquisas-demograficas`. Acessado: 2016-04-10.

Brucker, P., Heitmann, S. e Hurink, J. (2003). Flow-shop problems with intermediate buffers. *OR Spectrum*, 25(4):549–574.

Carlier, J. (1978). Ordonnancements a contraintes disjonctives. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 12(4):333–350.

Carlier, J. e Pinson, E. (1990). A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26.

Danna, E., Rothberg, E. e Le Pape, C. (2005). Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90.

Fischetti, M. e Monaci, M. (2014). Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731.

Garey, M. R., Johnson, D. S. e Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129.

Liao, C., Liao, L. e Tseng, C. (2006). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44(20):4297–4309.

Lin, S.-W. e Ying, K.-C. (2009). Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Production Research*, 47(5):1411–1424.

Potts, C. N. e Strusevich, V. A. (2009). Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, p. S41–S68.

Rossi, A. e Lanzetta, M. (2013). Scheduling flow lines with buffers by ant colony digraph. *Expert Systems with Applications*, 40(9):3328–3340.

Ruiz, R. e Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285.

Taillard, E. (2004). Scheduling instances. `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`. Acessado: 2016-04-05.

Tandon, M., Cummings, P. e LeVan, M. (1991). Flowshop sequencing with non-permutation schedules. *Computers & chemical engineering*, 15(8):601–607.

Yagmahan, B. e Yenisey, M. M. (2010). A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37(2):1361–1368.

Ying, K.-C. (2008). Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *The International Journal of Advanced Manufacturing Technology*, 38(3-4):348–354.

Capítulo 3

Dois algoritmos híbridos para o problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH)

Artigo científico a ser submetido para a revista Computers & Industrial Engineering, Elsevier.

Dois algoritmos híbridos para o problema de flow shop scheduling não permutacional com trabalhadores heterogêneos (FSNPTH)

Matheus de Freitas Araujo[a], José Elias C. Arroyo[a]

[a]*Departamento de Ciência da Computação, Universidade Federal de Viçosa, MG 36570-900, Brasil*

Resumo

O problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH) é classificado como um problema multicomponente, composto por dois problemas: O problema de alocação de trabalhadores em máquinas e o problema de *flow shop scheduling* não permutacional (FSNP). O objetivo do FSNPTH é minimizar o tempo máximo de conclusão das tarefas (*makespan*). A alocação de trabalhadores às máquinas é responsável pela definição dos tempos de processamentos das tarefas do problema de *flow shop scheduling* não permutacional. Para solucionar esse problema é proposto uma heurística para construção da solução inicial e dois algoritmos híbridos: VNS-IG que combinam as meta-heurísticas *Variable Neighborhood Search* (VNS) e *Iterated Greedy* (IG) e TS-IG que combinam as meta-heurísticas *Tabu Search* (TS) e IG. Os resultados obtidos pelos algoritmos propostos são comparados com a meta-heristica *Scatter Search* (SS) e com a resolução do modelo matemático pelo solver IBM ILOG CPLEX. Os resultados computacionais comprovam a eficiência do algoritmo proposto sobre os demais métodos em relação ao tempo de execução e qualidade das soluções obtidas.

Palavras-chaves: Flow Shop Scheduling, Algoritmos Híbridos, Problemas Multicomponente

1 Introdução

Devido a globalização e o crescimento econômico mundial, a necessidade por maior eficiência produtiva e melhor alocação de recursos disponíveis nas industrias tornou-se extremamente importante não só para a sobrevivência mas para torna-las competitivas no mercado. Com isso a programação (*scheduling*) da produção tornou essencial nas tomadas de decisões e aperfeiçoamento dos processos de produção. A

Email addresses: matheus.f.freitas@ufv.br (Matheus de Freitas Araujo), jarroyo@dpi.ufv.br (José Elias C. Arroyo)

programação da produção pode ser definido como uma das principais atividade do processo de produção de bens manufaturados. A programação da produção trata da alocação de recursos a tarefas em determinados períodos de tempo e seu objetivo é otimizar um ou mais objetivos. Os problemas de programação da produção são classificados de acordo com as características das máquinas e tarefas, os critérios de otimização, o ambiente das máquinas (simples, paralelas, *flow shop*, entre outras) e padrão de fluxo no sistema de produção, isso define um número ilimitado de problemas [1]. Esses problemas surgem com o objetivo de adequar a realidade de empresas e industrias de diversos de segmentos.

Dentre os problemas de programação da produção estão os problemas de *flow shop scheduling*. Esse problema surgiu em 1954 proposto por Johnson [2] e até hoje é um problema amplamente estudado por diversas industrias e cientistas. Problemas dessa natureza trazem um grande impacto no processo de produção uma vez que consiste em encontrar o sequenciamento de um conjunto de tarefas não preemptivas em um conjunto de máquinas dispostas em série, visando minimizar algumas medidas de desempenho, tal como, o tempo máximo de conclusão (*makespan*).

Os problemas de *flow shop scheduling* podem ser classificados em permutacional (FSP) e não permutacional (FSNP). No problema FSP objetivo é definir um único sequenciamento das tarefas. O sequenciamento define a ordem de execução das tarefas nas máquinas durante o processo de produção. Neste problema a quantidade de sequenciamentos de tarefas possíveis é $n!$, sendo n o número de tarefas. Nos problemas FSNP o sequenciamento das tarefas nas máquinas não é necessariamente a mesma, ou seja, cada máquina pode conter uma sequencia de processamento diferente. Nesse caso o numero de soluções possíveis é $(n!)^m$, onde m representa o numero de máquinas. Ambos os problemas são classificados como NP-Difícil para três ou mais máquinas [3].

Os problemas de *flow shop scheduling* não têm atendido completamente a realidade das empresas e industrias. Isso faz com que a combinação de diversos problemas de otimização combinatória seja cada vez mais comum para suprir as dificuldades enfrentadas no cotidiano das empresas e industrias. A abordagem apresentada por Benavides et al. [4] é um exemplo disso. Essa abordagem combina o problema FSNP com a designação de trabalhadores com diferentes habilidades em máquinas.

O problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH) proposto pro Benavides et al. [4], surgiu a partir da necessidade de alocação de trabalhadores deficientes com diferentes habilidades em postos de trabalhos, de forma que a heterogeneidade dos trabalhadores interfere no resultado final do sistema produtivo. Com isso esse problema visa melhorar o processo de produção, tornando competitivas as empresas que utilizam dessas mão de obra. Um exemplo desse tipo de industria são os Centros de Trabalho Protegido para Deficientes (CTDs), adotados com êxito em países como a Espanha. Os CTDs é uma das diversas iniciativas de integração de pessoas com deficiências no mercado de trabalho aplicada por diversos países. Nesses centros, grande parte ou até mesmo todos os trabalhadores sofrem com alguma deficiência (visual, auditiva, motora, mental ou

intelectual). O objetivo dos CTDs é introduzir e qualificar esses deficientes para que posteriormente possam ser absorvidas pelo mercado de trabalho (Chaves [5]). Iniciativas como essa são de extrema importância visto que os portadores de necessidades especiais sofrem com taxas de desempregos maiores que o restante da população.

Segundo Organização Internacional do Trabalho, pessoas com deficiência representam cerca de 10% da população mundial. Isso equivale a aproximadamente 700 milhões de pessoas no mundo, e aproximadamente 500 milhões estão em idade laboral. No Brasil, segundo o Instituto Brasileiro de Geografia e Estatística – IBGE, 23% da população brasileira sofre de alguma deficiência [6]. O IBGE ainda afirma que existem cerca de 23,7 milhões pessoas com deficiência em idade ativa desempregadas. Esses números, bem como a complexidade do problema FSNPTH faz com que esse problema torne-se especial, uma vez que se trata de um problema de difícil resolução de maneira exata e que pode impactar na qualidade do trabalho desempenhado pelas pessoas com deficiências.

Para solucionar o problema FSNPTH, inicialmente é proposto um algoritmo construtivo com objetivo de determinar uma solução inicial de boa qualidade. Em seguida são propostos dois algoritmos híbridos, denominados VNS-IG e TS-IG. O VNS-IG é baseada na combinação das meta-heurísticas *Variable Neighborhood Search* (VNS) e *Iterated Greedy* (IG). Já o TS-IG combina as meta-heurísticas *Tabu Search* (TS) e IG. Neste caso cada meta-heurística é utilizada para solucionar um subproblema especifico, o VNS e TS são utilizados para o solucionar o problema de designação de trabalhadores às máquinas e o IG responsável pela resolução do subproblema de FSNP.

A meta-heurística *Variable Neighborhood Search* (VNS), proposto por Mladenović e Hansen [7], busca melhorar a solução corrente explorando o espaço de busca do problema através de diversas estruturas de vizinhança, sendo assim, a solução corrente é atualizada sempre que uma solução vizinha melhor é encontrada. A utilização de diversas vizinhanças e a perturbação aplicada na solução corrente, dificulta que o VNS fique preso em ótimos locais durante a sua execução.

Originalmente a meta-heurística *Tabu Search* foi desenvolvida por Fred W. Glover em 1989 [8] e até hoje é utilizada para resolução de problemas de natureza combinatórial. Diversos trabalhos que possuem alocações de modo geral utilizam *Tabu Search*, como é o caso de Crainic et al. [9], Demir et al. [10] e Ren et al. [11].

A meta-heurística *Iterated Greedy* (IG) foi aplicado pela primeira vez em problemas de *flow shop scheduling* por Ruiz e Stützle [12]. Por se tratar de um algoritmo simples, com poucos parâmetros de controle, o IG vem sendo aplicado a diversos problemas de programação da produção. Um exemplo disso é o IG proposto por Benavides e Ritt [13] para resolução de problemas de *flow shop scheduling* não permutacional (FSNP).

Os resultados obtidos pelos algoritmos são comparados com os resultados obtidos pela meta-heurística *Scatter Search* desenvolvida por Benavides et al. [4]. Os algoritmos também são comparados com os resultados obtidos pela resolução do modelo matemático pelo solver IBM ILOG CPLEX.

O restante do artigo está organizado da seguinte forma. Na Seção 2 é apresentado uma revisão bibliográfica dos trabalhos relacionados ao problema de FSNPTH. Na Seção 3 é descrito formalmente o problema e definido o modelo matemático. Na Seção 4 são descritos os algoritmos propostos nesse artigo. Na Seção 5 são apresentadas as instâncias para o problema, o ambiente computacional e os parâmetros utilizados pelas meta-heurísticas. Nas Seções 6 e 7 são apresentados os resultados encontrados pelos métodos de resolução do problema, a conclusão desse artigo e as orientações futuras.

2 Revisão Bibliográfica

Um problema de *flow shop scheduling* pode ser classificando em permutacional (FSP) ou não permutacional (FSNP), a diferença entre os dois problemas é a complexidade de resolução e a quantidade de sequenciamentos possíveis [14]. A maior parte da literatura relacionada ao problema de *flow shop scheduling* apresenta a abordagem permutacional (FSP), para isso é apresentado uma breve visão geral para esse problema. Em seguida, é apresentado as principais abordagens relacionadas ao problema de FSNP e por fim os trabalhos similares ao do problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH).

O problema de *flow shop scheduling* permutacional consiste em definir uma única sequência de processamento de um conjunto de tarefas em máquinas não preemptivas dispostas em série, na qual, as máquinas compartilham da mesma sequência de processamento. O objetivo nesse problema é minimizar algumas medidas de desempenho, tais como, os tempos de entrega das tarefas (*completion time*) e/ou o tempo máximo de conclusão (*makespan*) do processo [15]. Diversos autores como Ruiz e Maroto [16], Potts e Trusevich [17] e Gupta e Stafford [18], definem o problema de FSP.

Os problemas de *flow shop scheduling* não permutacional (FSNP) difere do problema de FSP em um único ponto, quantidade de sequenciamentos necessário para construir a solução. Enquanto no problema de FSP busca encontra uma única sequência de processamento das tarefas que será aplicada em todas as máquinas, o problema de FSNP busca encontrar uma sequência de processamento para cada máquina do processo de produção. Isso faz com que os problemas de FSNP seja mais complexo. No problema de FSP o numero de sequenciamentos possíveis é definido por $n!$, onde n representa a quantidade de tarefas. No problema de FSNP a o a quantidade de sequenciamentos possíveis é de $(n!)^m$, onde m corresponde ao número de máquinas. Ambos os problemas são classificados como NP-Difícil para três ou mais máquinas [3].

Alguns autores propõem heurísticas construtivas para os problemas de *flow shop scheduling*, uma das principais heurísticas é NEH de Nawaz, Enscore, e Ham [19]. Essa heurística tem como objetivo definir o sequenciamento das tarefas do problema de FSP. O NEH consiste em alocar as tarefas às máquinas em duas etapas. Na Etapa 1, as tarefas são ordenadas de acordo com seus tempos de processamento.

Essa ordem de prioridade é usada para inserir as tarefas sucessivamente na fase de inserção da Etapa 2, na qual o sequenciamento final é construído. Na Etapa 2 cada tarefa é inserida na solução na posição do sequenciamento que minimize o *makespan* parcial da solução. Além do NEH, algumas outras heurísticas construtivas foram propostas, entretanto nenhuma obteve o mesmo desempenho do NEH como mostra Ruiz & Maroto [16].

Alguns autores aplicam meta-heurísticas para resolução do problema FSNP com objetivo de minimizar o valor do *makespan*. Tandon et al. [20] propõem a meta-heurística *Simulated Annealing* para FSNP. Os autores Brucker, Heitmann, & Hurink [21] e Liao et al. [22] propõem uma meta-heurística baseada em *Tabu Search* para encontrar boas soluções em tempo hábil. Além disso meta-heurísticas baseadas em Colônia de Formigas [23, 24] e *Iterated Greedy* [25] podem ser encontradas na literatura como opção de resolução desse problema.

Benavides e Ritt [13, 26] propõem duas heurísticas baseadas nos algoritmos *Iterated Local Search* (ILS) e *Iterated Greed* (IG) para resolver os problemas de FSP e FSNP respectivamente. Essas heurísticas se mostraram mais eficientes que as abordagens existentes na literatura para solucionar esses problemas.

O problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH) é um problema relativamente novo e não foram encontrados muitos trabalhos relacionados na literatura. Benavides et al. [4] define o problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos, propõem um modelo matemático para resolução de maneira exata do problema, e uma meta-heurística (*Scatter Search*) para encontrar soluções aproximadas. Araujo et al. [27] propõem utilização do método *Proximity Search*(PS) para resolver o problema. O PS é um método heurístico que utiliza o modelo matemático para encontrar as soluções ótimas do problema.

Alguns trabalhos [28, 29, 30, 31] apresentam problemas semelhantes ao FSNPTH. Carniel et al. [28] define uma variação do FSNPTH, na qual é possível definir a alocação de um ou dois trabalhadores às maquinas do sistema de produção *flow shop*. Em Benavides et al.[31] é definido um problema semelhante, entretanto utilizando problema de designação de trabalhadores em estações de trabalhos em um sistema de produção de *job shop*. Mencia et al. [30] aborda um problema de programação da produção sob a assistência de operadores humanos, nesse problema é considerado as relações de precedência nas tarefas, capacidade das máquinas e operadores e as habilidades dos trabalhadores durante o processamento de produção. Além desses trabalhos, Moreira, M. C. O. [29] propõem métodos para resolução do problema de balanceamento de linhas de produção com integração de trabalhadores com habilidades heterogêneas.

3 Definição do problema

O problema FSNPTH definido por Benavides et al. [4], consiste em alocar trabalhadores heterogêneos às máquinas e sequenciar tarefas nas máquinas em série de

forma que o *makespan* seja minimizado. A heterogeneidade dos trabalhadores é definida em relação ao tempo que cada trabalhador gasta para operar uma determinada máquina

Neste problema, existe um conjunto de trabalhadores $W = \{1, 2, ..., w\}$ que deve ser alocado em um conjunto $M = \{1, 2, ..., m\}$ de máquinas, sendo $w = m$, ou seja, cada trabalhador deve ser alocado em uma única máquina. Ao definir a alocação de trabalhadores às máquinas, tem-se o problema de *flow shop scheduling* não permutacional, onde existe um conjunto $N = \{1, 2, ..., n\}$ de tarefas que devem ser processadas no conjunto de máquinas (M) dispostas em série. Cada tarefa j é composta por um conjunto de m operações consecutivas $O_{1j}, O_{2j}, ..., O_{mj}$, onde O_{ij} é a operação da tarefa j na máquina i. O tempo de processamento de cada operação depende diretamente do trabalhador alocado à máquina, ou seja, se o trabalhador k é alocado à máquina i, o tempo da operação O_{ij} é t_{ijk}. Com isso, busca-se encontrar o melhor conjunto de operações que minimize o tempo de conclusão de todo o processo.

O problema FSNPTH é NP-Difícil, uma vez que o problema de *flow shop scheduling* não permutacional com $m \geq 3$ é classificado NP-Difícil [3]. Na Tabela 1 é apresentada uma instância para o problema de FSNPTH com 5 tarefas $(j_1, ..., j_5)$, 4 máquinas $(m_1, ..., m_4)$ e 4 trabalhadores $(w_1, ..., w_4)$, no qual, os valores representam o tempo de processamento de cada tarefa j em cada máquina i dado um trabalhador k (t_{ijk}). Os valores de $t_{ijk} = inf$ indicam que o trabalhador k não pode ser alocado à máquina i, ou seja, define uma impartibilidade entre o trabalhador k e a máquinas i.

Tabela 1: Uma instância do problema de FSNPTH.

Tarefas	m_1				m_2				m_3				m_4			
	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4
j_1	1	2	3	1	2	4	2	inf	2	4	2	2	2	2	2	2
j_2	1	1	4	3	1	1	4	inf	2	2	2	4	2	1	3	4
j_3	2	1	2	3	2	2	1	inf	2	1	2	4	4	2	2	3
j_4	2	5	2	3	1	4	2	inf	1	4	2	2	2	3	2	1
j_5	1	2	3	4	1	2	1	inf	2	2	2	3	2	1	3	2

Na Figura 1 é apresentada a solução ótima para a instância da Tabela 1. A permutação $P = \{2, 1, 3, 4\}$ representa designação dos trabalhadores às máquinas, sendo assim, o trabalhador 1 (w_1) é alocado a máquina 2 (m_2), o trabalhador 2 (w_2) é alocado a máquina 1 (m_1), o trabalhador 3 (w_3) é alocado a máquina 3 (m_3) e por fim o trabalhador 4 (w_4) alocado a máquina 4 (m_4). Além disso é apresentado o conjunto π de 4 sequências $(\{\pi_1, \pi_2, \pi_3, \pi_4\})$ representando uma solução com *makespan* igual a 16 ($C_{max} = 16$). As sequências $\pi_1 = \{2, 1, 5, 3, 4\}$, $\pi_2 = \{2, 1, 3, 5, 4\}$, $\pi_3 = \{2, 1, 3, 5, 4\}$ e $\pi_4 = \{2, 1, 3, 5, 4\}$ representam o sequenciamento das tarefas nas máquinas 1, 2, 3 e 4 respectivamente. Observa-se que, mesmo a designação não sendo ótima, levando em consideração a média ou a soma dos tempos de processamentos das tarefas, o valor do *makespan* (tempo de conclusão da última tarefa no

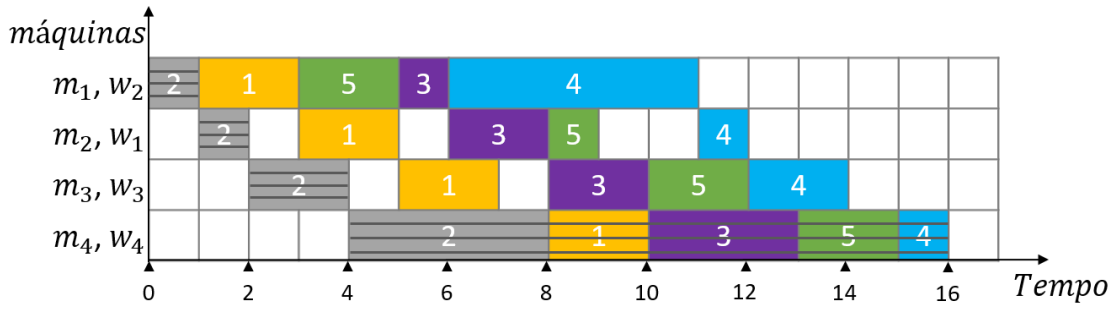


Figura 1: Solução ótima para a instância acima.

processo) é mínimo para este sequenciamento. Isso acontece por se tratar de um problema multicomponente, ou seja, um problema que consiste em dois ou mais problemas individuais [32]. Em problemas dessa natureza duas características podem ser observadas: a combinação e a interdependência. A combinação é a característica que determina que um problema seja a combinação de mais de um problema, já a interdependência faz com que os resultados dos problemas interfiram uns dos outros.

3.1 Modelagem Matemática de Programação Inteira Mista

Benavides et al. [4] propõem uma modelagem matemática para o problema FSNPTH na qual existem m máquinas, w trabalhadores e n tarefas, onde $m = w$. O modelo apresentado a seguir utiliza o seguinte parâmetro:

- G é um valor suficientemente grande definido por $\sum_{i=1}^{m}\sum_{j=1}^{n} max(t_{ijk} \forall k \in W)$.

O modelo utiliza as seguintes variáveis de decisão:

- C_{max}: Tempo máximo de conclusão das tarefas ou *makespan*;
- x_{ij}: Tempo de início da tarefa j na máquina i;
- p_{ij}: Tempo de processamento da tarefa j na máquina i;
- z_{ik}: Define a alocação do trabalhador k na máquina i, sendo $z_{ik} = 1$ se o trabalhador k é alocado a máquina i e $z_{ik} = 0$ caso contrário;
- $y_{ijj'}$: Define a precedência das tarefas, sendo $y_{ijj'} = 1$ se a tarefa j precede a tarefa j' na máquina i e $y_{ijj'} = 0$ caso contrário;

O modelo é apresentado a seguir:

$$\min C_{max} \tag{1}$$

sujeito a:

$$x_{mj} + p_{mj} \leq C_{max}, \forall j \in N \tag{2}$$

$$x_{ij} + p_{ij} \leq x_{i+1j}, \forall i \in M - 1, \forall j \in N \tag{3}$$

$$x_{ij} + p_{ij} \leq x_{ij'} + G(1 - y_{ijj'}), \forall i \in M, \forall j \neq j' \in N \tag{4}$$

$$y_{ijj'} + y_{ij'j} = 1, \forall i \in M, \forall j \neq j' \in N \tag{5}$$

$$p_{ij} = \sum_{k=1}^{w} t_{ijk} z_{ik} \forall i \in M, \forall j \in N \tag{6}$$

$$\sum_{k=1}^{w} z_{ik} = 1, \forall i \in M \tag{7}$$

$$\sum_{i=1}^{m} z_{iw} = 1, \forall w \in W \tag{8}$$

$$x_{ij} \geq 0, \forall i \in M, \forall j \in N \tag{9}$$

$$y_{ijj'} \in \{0, 1\}, \forall i \in M, \forall j \neq j' \in N \tag{10}$$

$$z_{ik} \in \{0, 1\}, \forall i \in M, \forall k \in W \tag{11}$$

A função objetivo, definida pela Equação 1, é minimizar o valor do *makespan* (C_{max}). A restrição 2 determina o valor de C_{max}. O conjunto de restrições 3 e 4 dizem respeito ao tempo de início de cada tarefa em cada máquina de acordo com as precedências das tarefas. A restrição de precedência das tarefas é definida pela Equação 5. A restrição 6 define o valor de p_{ij} a partir da alocação dos trabalhadores. As restrições 7 e 8 garante a designação dos trabalhadores as máquinas. A Equação 9 define a restrição de não negatividade das variáveis x_{ij}. O conjunto de restrições definidas pelas Equações 10 e 11 definem $y_{ijj'}$ e z_{ik} como variáveis binárias.

4 Meta-Heurísticas Propostas

O problema de FSNPTH é classificado como NP-Difícil, uma vez que o problema de FSNP é NP-Difícil para três ou mais máquinas [3]. Uma alternativa para tratar esse problema é utilizar algoritmos baseados em heurísticas e meta-heurísticas. Essas meta-heurísticas geralmente encontram boas soluções em tempo computacional aceitável, contudo não é possível provar a otimalidade dessas soluções.

Por se tratar de um problema multicomponente, combinar diferentes meta-heurísticas pode tornar mais simples a resolução do problema, uma vez que cada meta-heurística é responsável por resolver um subproblema específico. Partido desse principio, dois

algoritmos híbridos são propostos, VNS-IG, que se baseia nas meta-heurísticas *Variable Neighborhood Search* (VNS) e *Iterated Greedy* (IG), e TS-IG, que baseia-se nas meta-heurísticas *Tabu Search* (TS) e IG.

Nessa seção, primeiramente é apresentado a representação da solução e uma heurística para gerar a solução inicial (4.1). Em seguida é apresentado o algoritmo hibrido VNS-IG (4.2) e na Subseção 4.3 é apresentado o algoritmo híbrido TS-IG.

4.1 Representação da Solução e Solução Inicial

O problema de FSNPTH é um problema multicomponente que combina o problema de designação de trabalhadores em máquinas com o problema de FSNP. A solução para o problema é composta de duas partes. A primeira representa a designação dos trabalhadores às máquinas e a segunda representa os sequenciamentos das tarefas nas máquinas do sistema de produção. A designação dos trabalhadores as máquinas é representada por uma permutação P de w trabalhadores. O primeiro trabalhador da permutação será designado a máquina 1, o segundo trabalhador à máquina 2 e assim sucessivamente, ou seja, a posição i de P corresponde ao trabalhador alocado à máquina i. Por exemplo, uma permutação $P = \{w_5, w_1, w_4, w_3, w_2\}$ indica que o trabalhador w_5 está alocado à máquina m_1, o trabalhador w_1 à máquina m_2, o trabalhador w_4 à máquina m_3, o trabalhador w_3 à máquina m_4 e por fim, o trabalhador w_2 à máquina m_5.

O sequenciamento das tarefas nas máquinas é definido por conjunto $\pi = \{\pi_1, \pi_2, ..., \pi_m\}$ com m sequências (um para cada máquina), onde cada e elemento $\pi_i \in \pi$ define a ordem de processamento das n tarefas na máquina i. Então uma solução S do problema é representado por $S = (P, \pi)$. Onde $S(P)$ e $S(\pi)$ representam a permutação de trabalhadores e o sequenciamento das tarefas na solução S, respectivamente.

Nos algoritmos híbridos, VNS-IG e TS-IG, bem como na resolução do modelo matemático pelo solver IBM ILOG CPLEX, é utilizada uma solução inicial do problema de FSNPTH.

Por se tratar de um problema multicomponente, a construção da solução inicial é dividida em duas partes. A primeira parte define a solução para o problema de designação de trabalhadores às máquinas. A segunda define o sequenciamento das tarefas às máquinas do problema de FSNP ou seja, define o conjunto de sequencias de tarefas para as máquinas.

A primeira parte é baseada na heurística do custo mínimo, similar à heurística gulosa utilizada em problemas de transporte. Essa etapa do algoritmo consiste em alocar os trabalhadores as máquinas de forma que minimize os tempos médios de processamento das tarefas às máquinas.

Essa etapa do algorítimo define a alocação dos trabalhadores às máquinas a partir da média dos tempos de processamentos (t_{ijk}) das tarefas. Esse algoritmo utiliza uma abordagem gulosa, ou seja, a cada iteração do algoritmo a melhor alocação é definida.

O algoritmo de geração da solução inicial é aprestando pelo Algoritmo 1. O primeiro passo do algoritmo é determinar as médias dos tempos de processamento

trabalhadores-máquinas (T_{ik}), em seguida é determinado o conjunto I, contendo as máquinas que não podem ser operada por algum trabalhador (máquinas que possuem incompatibilidade) e atualiza o conjunto total de máquinas $M = M - I$. Em seguida, os trabalhadores são alocados às máquinas do conjunto I de forma que, a média de tempo de processamento seja o menor possível. A cada alocação trabalhador-máquina os conjuntos W e I são atualizados removendo o trabalhador e máquina alocados. O próximo passo é alocar trabalhadores às demais máquinas do conjunto M, seguindo a mesma lógica.

Algoritmo 1 Solução inicial para o problema FSNPTH.

1: **função** SOLUÇÃO_INICIAL( )
2: $T_{ik} = (\sum_{j \in N} t_{ijk}/n), \forall i \in M, \forall k \in W$
3: Determine o conjunto I com as máquinas que possuem incompatibilidades.
4: $M \leftarrow M - I$;
5: **enquanto** $I \neq \emptyset$ **faça**
6: $T'_{hg} \leftarrow min\{T_{ik} : i \in I, k \in W\}$;
7: Aloque o trabalhador g na máquina h em $S(P)$;
8: $I \leftarrow I - \{h\}$;
9: $W \leftarrow W - \{g\}$;
10: **fim enquanto**
11: **enquanto** $M \neq \emptyset$ **faça**
12: $T'_{hg} \leftarrow min\{T_{ik} : i \in M, k \in W\}$;
13: Aloque o trabalhador g na máquina h em $S(P)$;
14: $M \leftarrow M - \{h\}$;
15: $W \leftarrow W - \{g\}$;
16: **fim enquanto**
17: ALOCA_TAREFAS(S);
18: **retorna** S;
19: **fim função**

Uma vez definida a alocação dos trabalhadores a função ALOCA_TAREFAS determina a alocação das tarefas às máquinas, ou seja, o próximo passo é definir um sequenciamento de tarefas para cada máquina à partir da designação dos trabalhadores às máquinas. Na construção do sequenciamento das tarefas são utilizados os tempos de processamento determinados pela designação dos trabalhadores às máquinas. As melhores heurísticas construtivas para o problema de *flow shop scheduling* buscam alocar as tarefas inserindo uma tarefa por vez de maneira que minimize o *makespan* parcial a cada inserção. Neste trabalho é proposto uma variação do algoritmo NEH e é apresentado pelo Algoritmo 2.

O objetivo do Algoritmo 2 é definir dois sequenciamentos, π' e π''. O sequenciamento π' é utilizado nas duas primeiras maquinas (m_1 e m_2) e π'' é utilizado nas demais máquinas (m_i com $i > 2$). Em Benavides et al. [26] é utilizado 3 opções de inserção de tarefas nas máquinas para definir uma solução de FSNP. Entretanto

para utilizar o algoritmo FNS de Benavides [26] é necessário definir a ordem em que as tarefas serão inseridas na solução. O algoritmo proposto utiliza uma abordagem gulosa para definir a tarefa a ser inserida na sequência. A cada passo, todas as tarefas ainda não inseridas são testadas a fim de definir a tarefa j^* que ao ser inserida minimize o *makespan* parcial da solução. Para isso são considerado três opções de inserções em π' e π'' (como em Benavides [26]):

1. - Insere a tarefa j na posição p de π' e na posição $p-1$ de π'', $\forall p$, $2 \leq p \leq n$.
2. - Insere a tarefa j na posição p de π' e na posição $p+1$ de π'', $\forall p$, $1 \leq p \leq n-1$.
3. - Insere a tarefa j na posição p de π' e π''.

O primeiro passo do algoritmo consiste em determinar a tarefa $j \in N$ para ser inserida na primeira posição de π' e π'' e produza o menor *makespan* parcial em S . Ao alocar a tarefa, o conjunto de tarefas N é atualizado, removendo de N a tarefa inserida em π' e π''. A partir disso, o algoritmo avalia a inserção de cada tarefa $j \in N$ em cada posição $p \in \pi'$ levando em consideração as três opções apresentadas anteriormente. Uma vez definida a melhor tarefa j^* a ser inserida e a melhor posição p de inserção, a tarefa j^* é alocada em π' e π'' e a tarefa j^* é removida do conjunto N. Esse processo se repete até que todas as tarefas sejam alocadas, ou até que $N = \emptyset$. Ao final do algoritmo obtém-se π' e π'', que forma uma solução completa S do problema de FSNPTH, onde $S(\pi) = \{\pi', \pi', \pi'', ..., \pi''\}$.

Algoritmo 2 Determina o sequenciamento das tarefas para o problema de FSNPTH.

função ALOCA_TAREFAS(S)
  Determine a melhor tarefa j^* para ocupar a posição 1 de $S(\pi')$ e $S(\pi'')$;
  $S' \leftarrow S$; $N \leftarrow N - \{j^*\}$; $n' \leftarrow 1$;
  enquanto $N \neq \emptyset$ **faça**
    $f(S') \leftarrow \infty$;
    para cada $j \in N$ **faça**
      para $p \leftarrow 1$ **até** $n' + 1$ **faça**
        se $p > 1$ **então**
          Avalie a inserção de j em p de $S(\pi')$ e $p - 1$ de $S(\pi'')$;
        fim se
        se $p \leq n'$ **então**
          Avalie a inserção de j em p de $S(\pi')$ e $p + 1$ de $S(\pi'')$;
        fim se
        Avalie a inserção da tarefa j na posição p em $S(\pi')$ e $S(\pi'')$;
        se alguma inserção tiver um makespan menor que $f(S')$ **então**
          Insere a tarefa j em $S'(\pi')$ e $S'(\pi'')$;
          $j^* \leftarrow j$;
        fim se
      fim para
    fim para
    $S \leftarrow S'$; $n' \leftarrow n' + 1$; $N \leftarrow N - \{j^*\}$;
  fim enquanto
  retorna S
fim função

4.2 VNS-IG

O algoritmo VNS-IG proposto neste trabalho, combina as meta-heurísticas VNS e IG para resolver o problema de FSNPTH. A meta-heurística VNS é responsável por definir a alocação dos trabalhadores de forma que os tempos de processamento das tarefas melhorem o valor do *makespan*. Já o IG é responsável por resolver o subproblema de FSNP. No VNS-IG o valor do *makespan* é utilizado para guiar a busca. O VNS-IG é apresentado pelo Algoritmo 3.

O VNS-IG proposto inicia com a construção de uma solução inicial do problema de FSNPTH, nesse primeiro instante a melhor solução encontrada até o momento (S_{best}) é a solução inicial. Em seguida o algoritmo entra em um loop até que o tempo máximo de execução do algoritmo seja atingido. A partir desse passo, é aplicado o IG para melhorar o sequenciamento das tarefas na solução corrente (S). Feito isso parte-se para explorar as vizinhanças de S até que não seja encontrada uma solução melhor que a solução corrente. Logo, para cada estrutura de vizinhança (N_1 e N_2) são geradas as soluções vizinhas a S e a melhor solução dentre os vizinhos é armazenada em S'. Posteriormente o sequenciamento das tarefas em S' é melhorado

pelo IG. Caso o valor do *makespan* de S' seja menor que o valor do *makespan* de S, a solução corrente S é atualizada e então o VNS volta a explorar a primeira estrutura de vizinhança. Após explorar todas as vizinhanças e não encontrar uma solução S' melhor que S, S é comparado com S_{best} e caso necessário a solução S_{best} é atualizada. Na sequência, a pertubação é aplicada na designação dos trabalhadores as máquinas de S. Ao final do algoritmo a melhor solução (S_{best}) encontrada durante toda a execução do VNS-IG é retornada.

Algoritmo 3 VNS-IG para o problema de FSNPTH.

```
1: função VNS-IG(β, α, d, K)
2:     S ← Solução_Inicial();
3:     S_best ← S;
4:     enquanto Critério de parada Não é satisfeito faça
5:         S←IG(S, d, α);
6:         k ← 1;
7:         enquanto k ≤ K faça
8:             Encontre o melhor vizinho S' de S em N_k(S);
9:             S' ← IG(S', d, α);
10:            se f(S') < f(S) então
11:                S ← S';
12:                k ← 1;
13:            senão
14:                k ← k + 1;
15:            fim se
16:        fim enquanto
17:        se f(S) < f(S_best) então
18:            S_best ← S;
19:        fim se
20:        Perturbe solução S;
21:    fim enquanto
22:    retorna S_best;
23: fim função
```

A pertubação da solução aplicado pelo VNS é definida em 4.2.1, as estruturas de vizinhanças utilizadas pelo VNS são definidas na Subseção 4.2.2 e a meta-heurística IGA é apresentada na Subseção 4.2.3.

4.2.1 Pertubação

A pertubação usada no VNS consiste em alterar a permutação dos trabalhadores e pode ser descrita em dois passos. O primeiro passo consiste em remover aleatoriamente um trabalhador de sua máquina. O segundo passo consiste em inserir aleatoriamente o trabalhador removido na permutação. A inserção é feita utilizando

o conceito de inserção, seguindo a mesma lógica utilizada pela vizinhança N_2 (seção 4.2.2). Esses dois passos são repetidos β vezes, onde β é uma porcentagem do número de trabalhadores, ou seja, β remoções e inserção são realizadas.

4.2.2 Estruturas de Vizinhanças

No VNS são usadas duas estruturas de vizinhança: N_1 e N_2. A vizinhança N_1 é baseada na troca de dois trabalhadores na permutação de trabalhadores, e a vizinhança N_2 é baseada na remoção e inserção de um trabalhador na permutação dos trabalhadores.

A vizinhança N_1 consiste em trocar a posição de dois trabalhadores na permutação. O principal detalhe da estrutura N_1 é que em apenas duas máquinas os tempos de processamento das tarefas são alterados. O número máximo de vizinhos dessa estrutura é definida por $((w-1)*w)/2$, onde w representa a quantidade de trabalhadores, entretanto com a característica de incompatibilidade entre trabalhadores e máquinas o número de vizinhos pode ser menor para algumas instâncias. Suponha uma permutação com $w = 5$ e $m = 5$: $[w_1, w_2, w_3, w_4, w_5]$. Ao trocar os trabalhadores w_1 com w_3 a nova permutação será $[w_3, w_2, w_1, w_4, w_5]$, ou seja, os trabalhadores w_1 com w_3 são trocados de máquinas.

A segunda estrutura de vizinhança, N_2, é baseada na inserção de trabalhadores às máquinas. Neste caso, um trabalhador é retirado da máquina na qual está alocado e reinserido em outra máquina, fazendo com que boa parte ou até mesmo todos os trabalhadores sejam trocados de máquinas.

Novamente, suponha a seguinte permutação com $w = 5$ e $m = 5$: $[w_1, w_2, w_3, w_4, w_5]$. Ao inserir o trabalhador w_1 na posição 3 a nova permutação será $[w_2, w_3, w_1, w_4, w_5]$, note que além do trabalhador w_1, os trabalhadores w_2 e w_3 foram trocados de máquinas, isso permite encontrar soluções mais distantes da solução corrente no espaço de busca. O número máximo de vizinhos gerados é $(w-1)^2$, entretanto assim como na estrutura N_1, esse valor pode variar de acordo com a instância, uma vez que para algumas instancias existem trabalhadores que não podem ser designado a alguma(s) máquina(s).

4.2.3 Iterated Greedy

O *Iterated Greedy* (IG) é um algoritmo simples e pode ser divido em quatro fases principais. A primeira fase consiste em remover aleatoriamente algumas tarefas da solução, obtendo assim, uma solução parcial do problema FSNP. Em seguida essas tarefas são reinseridas na solução com objetivo de melhorar o valor do *makespan*. A terceira fase consiste em aplicar uma busca local a partir da solução gerada na fase anterior. O objetivo dessa fase é localizar uma solução próxima à solução atual com um valor de *makespan* menor. Na quarta e última fase, é aplicado um critério de aceitação que define se a solução corrente deve ser substituída pela a solução encontrado durante a busca local. O *Iterated Greedy* proposto é apresentado pelo Algoritmo 4 e utiliza os seguintes parâmetros:

1. - d: Quantidade de tarefas a serem removidas aleatoriamente e reinseridas na solução;
2. - α: Parâmetro utilizado no critério de aceitação.

Algoritmo 4 Iterated greedy para problemas de *flow shop scheduling* não permutacional.

1: **função** IG(S, d, α)
2: **enquanto** *critério de parada* não é satisfeito **faça**
3: Remova d tarefas aleatórias de cada sequência π_i de S para obter S' e o conjunto D tarefas removidas;
4: $n' \leftarrow n - d$;
5: **para cada** $j \in D$ **faça**
6: **para** $p \leftarrow 1$ **até** $n' + 1$ **faça**
7: **se** $p > 1$ **então**
8: Avalie a inserção da tarefa j na posição p em $S'(\pi_1), S'(\pi_2)$ e na posição $p-1$ em $S'(\pi_3), ..., S'(\pi_m)$;
9: **fim se**
10: **se** $p \leq n'$ **então**
11: Avalie a inserção da tarefa j na posição p em $S'(\pi_1), S'(\pi_2)$ e na posição $p+1$ em $S'(\pi_3), ..., S'(\pi_m)$;
12: **fim se**
13: Avalie a inserção da tarefa j na posição p em $S'(\pi_1), ..., S'(\pi_m)$;
14: **fim para**
15: Aplique a melhor inserção de j em S';
16: $n' \leftarrow n + 1$;
17: **fim para**
18: $S' \leftarrow$ Busca Local(S');
19: **se** $Rand(0, 1) < Prob$ **então**(eq. 12)
20: $S \leftarrow S'$;
21: **fim se**
22: **fim enquanto**
23: **retorna** S;
24: **fim função**

A destruição da solução funciona da seguinte maneira. Dada uma solução completa S, formado por m sequências $\pi_1, \pi_2, ..., \pi_m$, na qual cada sequência π_i representa o sequenciamento das tarefas na máquina i. Remove-se d tarefas aleatoriamente da solução e obtém-se um conjunto D com as tarefas removidas e uma solução parcial S' onde, cada sequência π_i contém $n-d$ tarefas. Em seguida, para cada tarefa $j \in D$ e para cada posição $p \in \pi_1$ é avaliado a inserção da tarefa j na posição p levando em consideração três critérios.

I - Insere a tarefa j na posição p de π_1 e π_2, e na posição $p-1$ de $\pi_3, ..., \pi_m$, $\forall p$, $2 \leq p \leq n$.

II - Insere a tarefa j na posição p das sequências $\pi_1, ..., \pi_m$.

III - Insere a tarefa j na posição p de π_1 e π_2, e na posição $p+1$ de $\pi_3, ..., \pi_m$, $\forall p$, $1 \leq p \leq n-1$.

Levando em consideração essas três opções de inserção (I, II e III), considera-se a melhor inserção da tarefa j, formando assim uma nova solução S'. Um exemplo dessas três opções de inserção é apresentado na Figura 2. Note que considerando a opção I, a Tarefa 1 é inserida na posição 2 nas máquinas m_1 e m_2 e na posição 1 nas máquinas m_3 e m_4. Considerando a opção II, a tarefa 1 é inserida na posição 2 em todas as máquinas, ou seja, nas máquinas m_1, m_2, m_3 e m_4. Considerando a opção III, a Tarefa 1 é inserida na posição 2 nas máquinas m_1 e m_2 e na posição 3 nas máquinas m_3 e m_4. Neste exemplo, para $p = 2$, são gerados três soluções. Das três soluções geradas, a melhor solução parcial é selecionada. Note que para $p = 1$ e $p = n - 1$, somente serão geradas duas soluções.

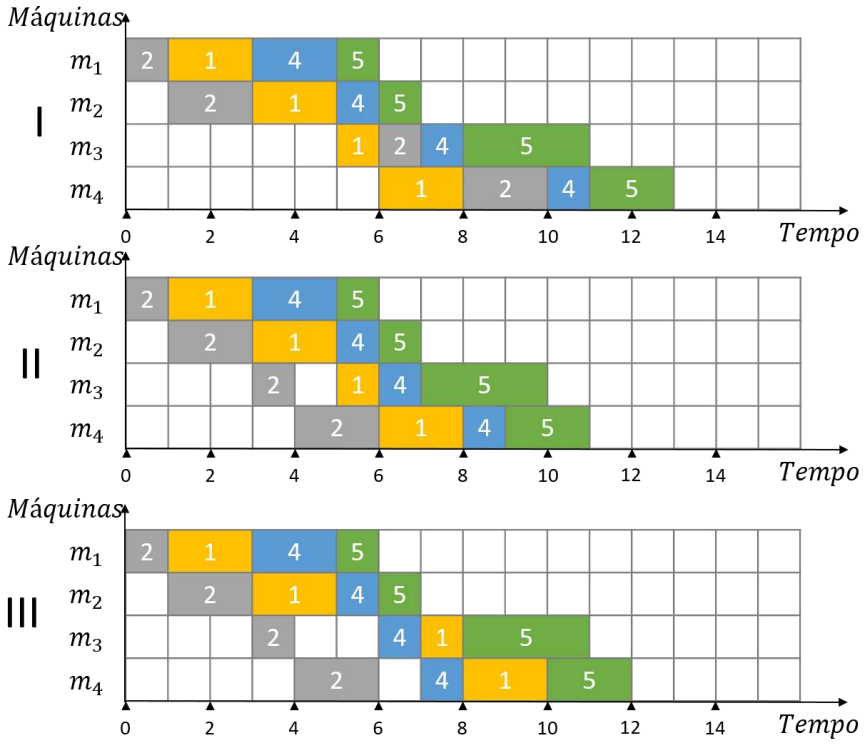


Figura 2: Exemplo de inserção da tarefa 1 na posição 2 da solução parcial levando em consideração as três opções apresentadas anteriormente (neste exemplo $p = 2$)

Ao obter a solução com o sequenciamento completo (S'), uma busca local *Best Improvement* é aplicada utilizando uma variação da vizinhança proposta por Nowicki

and Smutnicki [33]. Uma busca local *Best Improvement* consiste em passa repetidamente para o melhor vizinho, até que a solução seja um mínimo local. Essa vizinhança utiliza o conceito de caminho crítico para definir os vizinhos de uma solução. O caminho crítico de uma solução de FSNP é definido como sendo uma sequência de operações consecutivas na mesma máquina ou da mesma tarefa que leva à duração global mais longa. Um caminho crítico pode ser decomposto em blocos máximos de operações executados na mesma máquina (veja a Figura 1). O caminho crítico é representado pelas listas horizontais.

A vizinhança de Nowicki and Smutnicki limita-se a trocar apenas as duas primeiras ou as duas últimas tarefas consecutivas de um bloco no caminho crítico da solução. Neste trabalho é proposto uma variação dessa estrutura na qual a troca das tarefas é aplicada entre todas as tarefas do mesmo bloco. Por exemplo, na Figura 1, os pares de tarefas a serem trocadas no bloco 4 do caminho critico são: (2,1), (2,3), (2,5), (2,4), (1,3), (1,5), (1,4), (3,5), (3,4) e (5,4), gerando um total de 10 soluções vizinhas. Segundo a vizinhança de Nowicki and Smutnicki [33], as trocas realizadas seriam (2,1) e (5,4), gerando apenas 2 vizinhos.

Após a busca local, o critério de aceitação de *Metropolis* (definido pela Equação 12) é utilizado para definir a probabilidade de aceitação de solução S' como solução corrente.

$$Prob = min\{e^{-\Delta(s,s')/T}, 1\} \tag{12}$$

onde $\Delta(S, S')$ é a diferença do total de *makespan* entre as soluções, ou seja $\Delta(S, S') = f(S') - f(S)$ e a temperatura $T = \alpha\overline{p}n/10$, onde $\overline{p}$ é definido pela Equação 13, e representa o tempo médio de processamento de uma operação.

$$\overline{p} = \sum_{j \in N} \sum_{i \in M} p_{ij}/nm \tag{13}$$

O critério de parada utilizado pelo IG é o numero de iterações. Nessa trabalho foi o numero máximo de iterações foi fixado em n, que representa o número de tarefas da instância. Ao final do algoritmo a solução com menor *makespan* é retornada.

4.3 TS-IG

Nessa seção é apresentada o algoritmo híbrido TS-IG esse algoritmo utiliza como base a meta-heurística *Tabu Search* (TS), responsável pela resolução do subproblema de alocação dos trabalhadores às máquinas e a meta-heurística IG para encontrar os melhores sequenciamentos de tarefas.

O objetivo do TS nesse problema é determinar a permutação de trabalhadores a partir de uma busca local baseada na troca de posição entre os trabalhadores na permutação. O movimento tabu é definido por uma estrutura contendo um trabalhador (w_k) e uma máquina (m_i). Essa meta-heurística utiliza a vizinhança N_1 (definida na seção 4.2.2), logo ao trocar a posição de dois trabalhadores na permutação, são gerados dois movimentos proibidos correspondente à troca realizada.

Suponha que uma determinada solução S possua a seguinte permutação com $w = 5$ e $m = 5$: $[w_2, w_1, w_5, w_4, w_3]$ e o melhor vizinho a partir da vizinhança N_1 contém a permutação: $[w_5, w_1, w_2, w_4, w_3]$, ou seja, w_2 e w_5 foram trocados de posição. Os movimentos definidos por essa troca é dado por (w_2, m_1) e (w_5, m_3) e então inseridos na lista tabu. Com isso o algoritmo impede que o trabalhador w_2 volte à posição 1 e que o trabalhador w_5 volte à posição 3 da permutação por um determinado período de tempo. O tempo em que um movimento permanece como tabu está diretamente ligado ao tamanho da lista tabu (γ). A lista tabu é implementada como uma lista circular, logo um movimento deixa de ser tabu quando é substituído por outro movimento na lista.

Nessa meta-heurística existe apenas uma maneira da solução ser substituída por uma solução que contenha um movimento tabu, essa maneira é definida como critério de aspiração. O critério de aspiração é aplicado apenas se a solução é melhor que a melhor solução (S_{best}) encontrada até o momento pelo algoritmo. O Algoritmo 5 define os passos do TS-IG.

Algoritmo 5 TS-IG para o problema de FSNPTH.

```
1:  função TS-IG(d, α, γ)
2:      S ← Solução_Inicial();
3:      S ← IG(S, d, α);
4:      S_best ← S;
5:      enquanto critério de parada não é satisfeito faça
6:          f(S*) ← ∞;
7:          para cada S' ∈ N1(S) faça
8:              se S' não contém movimentos proibidos e f(S') < f(S*) então
9:                  S* ← S';
10:             senão se f(S') < f(S_best) então
11:                 S_best ← S';
12:             fim se
13:         fim para
14:         Atualiza a lista de tabu com S*;
15:         S ← IG(S*, d, α);
16:         se f(S) < f(S_best) então
17:             S_best ← S;
18:         fim se
19:     fim enquanto
20:     retorna S_best;
21: fim função
```

O algoritmo TS-IG recebe os parâmetros: d, α e γ . Inicialmente é gerada uma solução que posteriormente é melhorada pelo IG (definido na subseção 4.2.3). A solução retornada pelo IG e armazenada como melhor solução já encontrada (S_{best}). O próximo passo é iniciar um *loop* até que o tempo máximo de execução seja atingido.

Em seguida é os vizinhos de S são gerados e então verificado se existe alguma solução com *makespan* menor que da melhor solução encontrada até o momento S_{best} e também o melhor vizinho que não possua movimentos proibidos (S^*). Em seguida a lista tabu é atualizada com os movimentos que geraram S^*. No próximo passo, a solução corrente S é substituída pela solução retornada pela meta-heurística IG, que busca melhorar o sequenciamento das tarefas da solução S^*, consequentemente melhorando o valor do *makespan*. Por fim a solução corrente é armazenada em S_{best} caso o possua um *makespan* menor que a melhor solução encontrada até o momento. Ao final do algoritmo, a solução com menor valor de *makespan* (S_{best}) é retornada.

5 Experimentos Computacionais

Nessa seção inicialmente são descritas as instâncias utilizadas nos experimentos computacionais realizados com os algoritmos propostos. Em seguida são apresentados os experimentos relacionados à calibração dos parâmetros dos algoritmos. A calibração dos parâmetros é feita com o proposito de obter o melhor desempenho dos algoritmos.

5.1 Instâncias

Para realizar os experimentos computacionais e avaliar o desempenho das meta-heurísticas propostas, são utilizados dois grupos de instâncias propostas Benavides et al. [4]. O primeiro grupo é baseadas nas instâncias de Carlier [34] e são consideradas pequenas, com m e w variando de 5 a 9 e n variando de 7 a 14. O segundo grupo é baseado nas instâncias de Taillard [35], que são instâncias grandes, com m e w variando de 5 a 20 e n variando de 20 a 50.

Benavides et al. [4] assumem que o tempo de processamento p_{ij} das instâncias originais são de um trabalhador padrão. A partir disso, as instâncias são modificadas de duas maneiras: primeiro adiciona-se uma porcentagem fixa de incompatibilidade, que consiste em definir a quantidade de trabalhadores com limitações de executar operações em uma máquina, no exemplo apresentado na Tabela 1, a incompatibilidade se dá pelo trabalhador 4 na máquina 2. Os autores optaram por gerar instâncias com 0%, 10% e 20% de incompatibilidades. Em seguida, o tempo de processamento t_{ijk} de um trabalhador padrão para algumas operações são aumentados escolhendo aleatoriamente um tempo de processamento no intervalo $[p_{ij}; 2p_{ij}]$ ou $[p_{ij}; 5p_{ij}]$. Este procedimento é repetido w vezes para definir os tempos t_{ijk} dos w diferentes trabalhadores. Os valores de *ub* das instâncias de Carlier e Taillard para os problema de FSNPTH bem como as instâncias utilizadas nos experimentos são disponibilizadas pelos autores e podem ser encontradas em: http://www.inf.ufrgs.br/algopt/hetFS/.

5.2 Ambiente Computacional

Os experimentos computacionais foram realizados a fim de demonstrar o desempenho dos algoritmos híbridos proposto em relação a meta-heurística *Scatter Search* (SS) proposta por Benavides et al. [4], a resolução do modelo matemático a partir

de uma solução inicial e sem a solução inicial. Os algoritmos VNS-IG e TS-IG, bem como os demais algoritmos utilizados nesse trabalho foram desenvolvidas na linguagem C++. As implementações referente ao modelo matemático foi desenvolvido utilizando a biblioteca ILOG Concert e os modelos foram resolvidos pelo solver IBM ILOG CPLEX.

Os experimentos computacionais dos algoritmos propostos foram realizados em um computador Intel Core i7 6700K, CPU (4.0 GHz), com 32 GB de RAM, sistema operacional Windows 10 profissional. Os Algoritmos foram compilados pelo software Microsoft Visual Studio 2015 Community com *optimization level* 2 (-O2) e executados em *single thread*. Os experimentos computacionais do *Scatter Search* (SS) foram executados em um computador AMD Opteron, CPU (2.9 GHz), com 64 GB de RAM, sistema operacional Linux (Benavides et al. [4]). Os autores fixaram o tempo máximo de execução do SS em 600 segundos para cada instância. Os experimentos computacionais relacionados ao modelo matemático foram realizados em um *Cluster* onde cada nó contém 2 processadores Intel(R) Xeon(R) CPU X5650 (12M Cache, 2.66 GHz, 6.40 GT/s Intel(R) QPI, 6 *cores*, 12 *threads*) e 24 GB de RAM DDR3 1333 MHz. Entretanto os experimentos foram realizados utilizando apenas 1 processador, 4 *threads* e 8GB de memória RAM. Nas resoluções do modelo matemático o tempo de execução foi fixado em 3600 segundos para cada instância.

5.3 Calibração de Parâmetros das Meta-heurísticas

Esta seção define os valores dos parâmetros utilizados nas meta-heurísticas VNS-IG e TS-IG, além disso são apresentados os experimentos computacionais da calibração e os resultados.

5.3.1 VNS-IG

O *Iterated Greedy* utiliza dois parâmetros, d e α. Para testar o parâmetro d foi definido um conjunto de valores que variam de 2 a 6 ($\{2, 3, 4, 5, 6\}$) e o parâmetro α variando de 0,2 a 0,8, sendo o conjunto $\{0,2; 0,4; 0,6; 0,8\}$. Para o *Variable Neighborhood Search* o parâmetro a ser calibrado é o valor de β, que representa porcentagem de trabalhadores a serem removidos e realocado às máquinas a cada iteração do algoritmo. Os resultados preliminares definiram os valores de β variando de 10% a 30% ($\{10\%, 20\%, 30\%\}$).

Para calibrar o algoritmo VNS-IG foi utilizado a técnica de fatorial completo, ou seja, todos os parâmetros são combinados entre si. A quantidade de combinações de parâmetros para o VNS-IG é 60 ($3 * 5 * 4$, que representam β, d e α). Os experimentos e a combinação dos parâmetros são apresentados na Tabela 2 (os experimentos são representados pela coluna Exp.). Os testes foram realizados utilizando um conjunto de 72 instâncias baseadas em Taillard definidas aleatoriamente. Para cada combinação de parâmetros foram realizados 5 repetições.

Para avaliar os resultados dos experimentos, assim como Benavides [4], foram calculados os valores do Desvio Percentual Relativo (RPD, do inglês *Relative Percentage Deviation*) em relação ao *ub*. O RPD pode ser calculado através da Equação

Tabela 2: Combinações fatorial completa dos parâmetros utilizados pelo VNS-IG.

Exp.	d	α	β	Exp.	d	α	β	Exp.	d	α	β
1	2	0,2	0,1	21	3	0,6	0,3	41	5	0,4	0,2
2	2	0,2	0,2	22	3	0,8	0,1	42	5	0,4	0,3
3	2	0,2	0,3	23	3	0,8	0,2	43	5	0,6	0,1
4	2	0,4	0,1	24	3	0,8	0,3	44	5	0,6	0,2
5	2	0,4	0,2	25	4	0,2	0,1	45	5	0,6	0,3
6	2	0,4	0,3	26	4	0,2	0,2	46	5	0,8	0,1
7	2	0,6	0,1	27	4	0,2	0,3	47	5	0,8	0,2
8	2	0,6	0,2	28	4	0,4	0,1	48	5	0,8	0,3
9	2	0,6	0,3	29	4	0,4	0,2	49	6	0,2	0,1
10	2	0,8	0,1	30	4	0,4	0,3	50	6	0,2	0,2
11	2	0,8	0,2	31	4	0,6	0,1	51	6	0,2	0,3
12	2	0,8	0,3	32	4	0,6	0,2	52	6	0,4	0,1
13	3	0,2	0,1	33	4	0,6	0,3	53	6	0,4	0,2
14	3	0,2	0,2	34	4	0,8	0,1	54	6	0,4	0,3
15	3	0,2	0,3	35	4	0,8	0,2	55	6	0,6	0,1
16	3	0,4	0,1	36	4	0,8	0,3	56	6	0,6	0,2
17	3	0,4	0,2	37	5	0,2	0,1	57	6	0,6	0,3
18	3	0,4	0,3	38	5	0,2	0,2	58	6	0,8	0,1
19	3	0,6	0,1	39	5	0,2	0,3	59	6	0,8	0,2
20	3	0,6	0,2	40	5	0,4	0,1	60	6	0,8	0,3

14, onde $Makespan$ é o média do *makespan* encontrado pelos algoritmos VNS-IG e TS-IG em uma determinada configuração de parâmetros após as 5 repetições e ub é o melhor valor conhecido para uma determinada instância.

$$RPD = \frac{Makespan - ub}{ub} \times 100 \tag{14}$$

Para validar os resultados obtidos na calibração dos parâmetros dos algoritmos e verificar se as diferenças observadas são estatisticamente significantes, foi realizada uma Análise de Variância (ANOVA) [36]. Também foram verificadas as três pressuposições da ANOVA para que os resultados do teste sejam estatisticamente válidos: normalidade, igualdade de variância e a independência dos resíduos. Dado que o $valor - P$ na ANOVA resultou em $0,00$ para o VNS-IG e este valor é menor que $0,05$, pode-se concluir que estatisticamente existe diferenças significativas entre os experimentos.

Os gráficos de médias resultante do teste HSD de Tukey com nível de confiança de 95% para confrontar os experimentos do VNS-IG é apresentado na Figura 3. Como os intervalos de alguns experimentos se sobrepõem na Figura 3, o experimento com menor RPD médio é selecionado, logo a combinação de parâmetros do experimento 16 ($d = 3$, $\alpha = 0,4$ e $\beta = 0,1$) é utilizado no VNS-IG.

A Figura 5 apresenta os gráficos de médias e intervalos HSD de Tukey com nível de confiança de 95%, para os resultados do VNS-IG com os seguinte tempos de execução (em segundos): $0,1mn$, $0,2mn$ e $0,3mn$, onde m representa o total de máquinas e n representa o número de tarefas da instância. Observa-se que quanto

Tabela 3: Combinações fatorial completa dos parâmetros utilizados pelo TS-IG.

Exp.	d	α	γ	Exp.	d	α	γ	Exp.	d	α	γ
1	2	0,2	10%	34	3	0,6	40%	67	5	0,4	20%
2	2	0,2	20%	35	3	0,6	5%	68	5	0,4	30%
3	2	0,2	30%	36	3	0,8	10%	69	5	0,4	40%
4	2	0,2	40%	37	3	0,8	20%	70	5	0,4	5%
5	2	0,2	5%	38	3	0,8	30%	71	5	0,6	10%
6	2	0,4	10%	39	3	0,8	40%	72	5	0,6	20%
7	2	0,4	20%	40	3	0,8	5%	73	5	0,6	30%
8	2	0,4	30%	41	4	0,2	10%	74	5	0,6	40%
9	2	0,4	40%	42	4	0,2	20%	75	5	0,6	5%
10	2	0,4	5%	43	4	0,2	30%	76	5	0,8	10%
11	2	0,6	10%	44	4	0,2	40%	77	5	0,8	20%
12	2	0,6	20%	45	4	0,2	5%	78	5	0,8	30%
13	2	0,6	30%	46	4	0,4	10%	79	5	0,8	40%
14	2	0,6	40%	47	4	0,4	20%	80	5	0,8	5%
15	2	0,6	5%	48	4	0,4	30%	81	6	0,2	10%
16	2	0,8	10%	49	4	0,4	40%	82	6	0,2	20%
17	2	0,8	20%	50	4	0,4	5%	83	6	0,2	30%
18	2	0,8	30%	51	4	0,6	10%	84	6	0,2	40%
19	2	0,8	40%	52	4	0,6	20%	85	6	0,2	5%
20	2	0,8	5%	53	4	0,6	30%	86	6	0,4	10%
21	3	0,2	10%	54	4	0,6	40%	87	6	0,4	20%
22	3	0,2	20%	55	4	0,6	5%	88	6	0,4	30%
23	3	0,2	30%	56	4	0,8	10%	89	6	0,4	40%
24	3	0,2	40%	57	4	0,8	20%	90	6	0,4	5%
25	3	0,2	5%	58	4	0,8	30%	91	6	0,6	10%
26	3	0,4	10%	59	4	0,8	40%	92	6	0,6	20%
27	3	0,4	20%	60	4	0,8	5%	93	6	0,6	30%
28	3	0,4	30%	61	5	0,2	10%	94	6	0,6	40%
29	3	0,4	40%	62	5	0,2	20%	95	6	0,6	5%
30	3	0,4	5%	63	5	0,2	30%	96	6	0,8	10%
31	3	0,6	10%	64	5	0,2	40%	97	6	0,8	20%
32	3	0,6	20%	65	5	0,2	5%	98	6	0,8	30%
33	3	0,6	30%	66	5	0,4	10%	99	6	0,8	40%
								100	6	0,8	5%

maior o tempo de execução, menores os valores do RPD e consequentemente melhores as soluções encontradas. Nos resultados encontrado pelo VNS-IG não há diferenças significativas entre os tempos de execução, uma vez que o $valor - P$ na ANOVA resultou em $0,078$. Sendo assim, o tempo máximo de execução dos algoritmos foi fixado em $0,1mn$ segundos.

5.3.2 TS-IG

Para calibrar o TS-IG é preciso levar em consideração os parâmetros do *Tabu Search* (TS) e IG. O TS possui apenas um parâmetro a ser calibrado, que representa o tamanho da lista tabu (γ), os testes preliminares apontaram o γ variando entre $\{5\%, 10\%, 20\%, 30\%, 40\%\}$ da quantidade de vizinhos gerado pela vizinhança

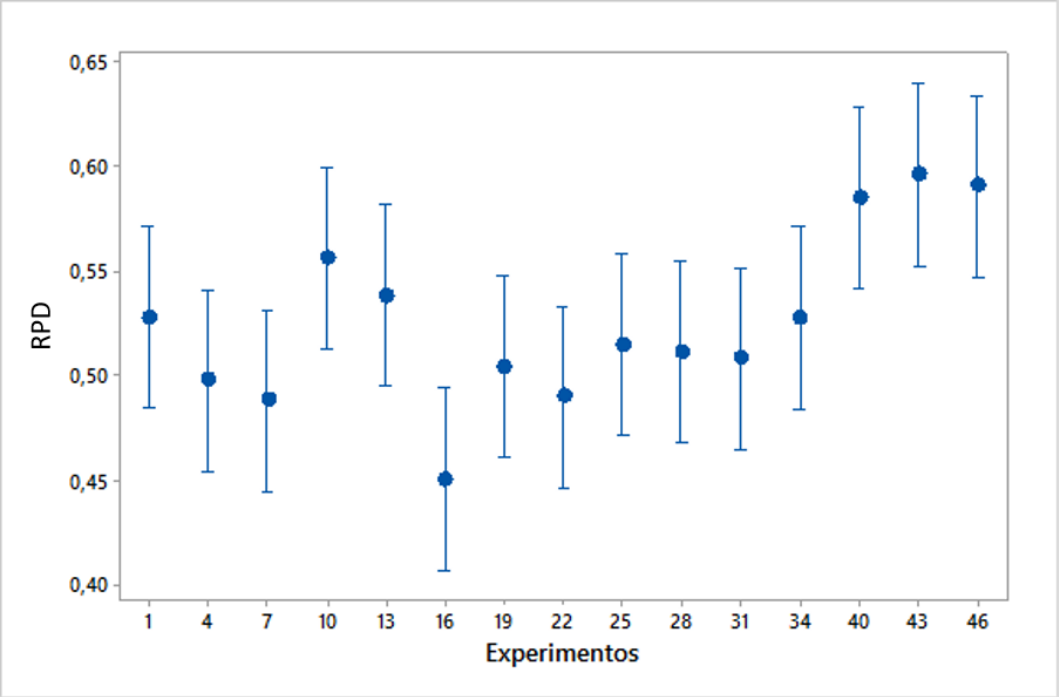


Figura 3: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos experimentos de calibração dos VNS-IG

Tabela 4: Tempo máximo de execução(em segundos) dos algoritmos VNS-IG e TS-IG.

Grupo das Instâncias	Tamanho		Tempo Max.
	n	m	
Instâncias de Carlier	11	5	5,5
	13	4	5,2
	12	5	6
	14	4	5,6
	10	6	6
	8	9	7,2
	8	8	4,9
	7	7	6,4
Instâncias de Taillard	20	5	10
	20	10	20
	20	20	40
	50	5	25
	50	10	50
	50	20	100

N_1 $(((w-1)*w)/2)$. Assim como no VNS-IG para calibrar o algoritmo TS-IG foi utilizado a técnica de fatorial completo. A quantidade de combinações dos parâmetros para o TS-IG é 100 $(5*5*4$, que representam γ, d e $\alpha)$. Os experimentos e as combinações dos parâmetros é apresentado na Tabela 3. Os testes foram realizados utilizando o mesmo conjunto de instâncias definido em 5.3.1 . Para cada combinação

de parâmetros foram realizados 5 repetições.

Assim como a calibração do VNS-IG são calculados os valores de RPD e realizado a Análise de Variância (ANOVA) [36] para verificar se existem diferença significativas dos experimentos. Dado que o $valor - P$ na ANOVA resultou em $0,00$ para o TS-IG e este valor é menor que $0,05$, pode-se concluir que estatisticamente existe diferenças significativas entre os experimentos.

Os gráficos de médias resultante do teste HSD de Tukey com nível de confiança de 95% para confrontar os experimentos do TS-IG é apresentado na Figura 4. Como os intervalos de alguns experimentos se sobrepõem na Figura 4, o experimento com menor RPD é selecionado, logo a combinação de parâmetros do experimento 30 é utilizado, esse experimento utiliza os parâmetros $d = 3$, $\alpha = 0,4$ e $\gamma = 5\%$ do número de vizinhos gerados por N_1.

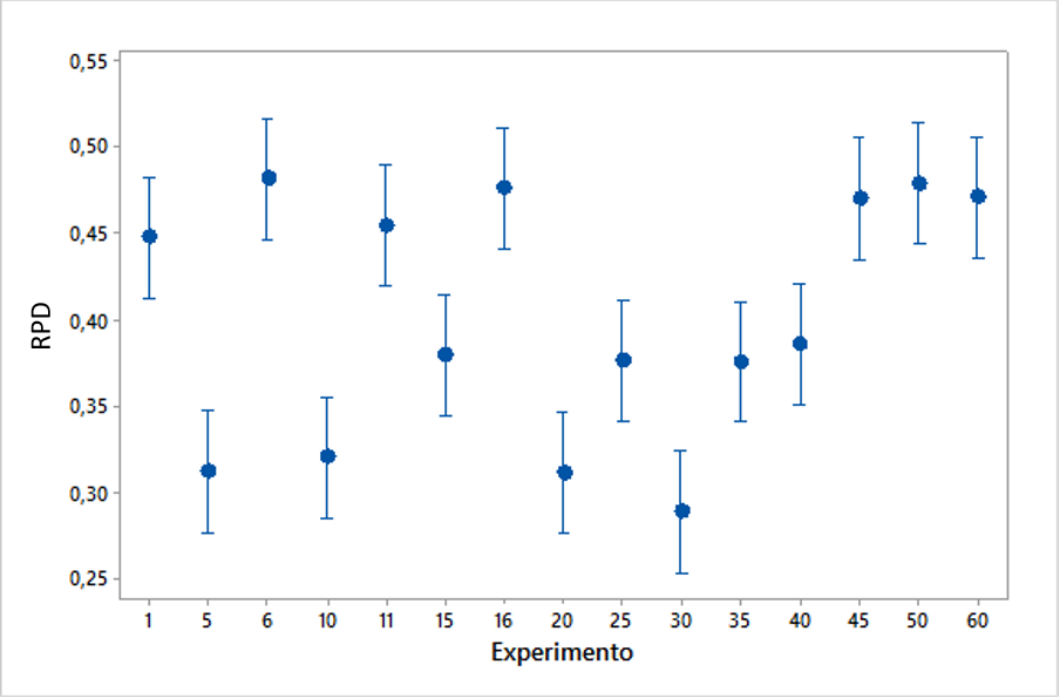


Figura 4: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos experimentos de calibração dos TS-IG

Assim como no VNS-IG, foram testados diferentes tempos de execução, a Figura 6 apresenta os gráficos de médias e intervalos HSD de Tukey com nível de confiança de 95%, para os resultados do TS-IG com os seguinte tempos de execução (em segundos): $0,1mn$, $0,2mn$ e $0,3mn$, onde m representa o total de máquinas e n representa o número de tarefas da instância. Nos resultados encontrado pelo TS-IG não há diferenças significativas entre os tempos de execução, uma vez que o $valor - P$ na ANOVA resultou em $0,078$. Sendo assim, o tempo máximo de execução dos algoritmos foi fixado em $0,1mn$ segundos.

Os tempos máximos de execução para cada instância a partir dos resultados dos testes é apresentado pela Tabela 4. Este trabalho não implementa o *Scatter Search*, uma vez que a diferença entre os tempos máximos de execução entre as meta-heurísticas propostas e do SS no pior caso é de 6 vezes menor e a diferença entre os processadores não chega a essa diferença. Segundo as especificações dos fabricantes, sites especializados e sites de *benchmarks* [37, 38, 39, 40, 41, 42, 43, 44], o processador Intel Core i7 6700K é superior apenas na velocidade de *clock* com uma diferença de 40% para o AMD Opteron utilizado por Benavides et al.[4]. Em todos os outros quesitos como, como quantidade de cores, tamanho e quantidade da memória cache L2, o processador AMD Opteron é melhor [45].

6 Resultados Computacionais

Os resultados obtidos pelo VNS-IG e TS-IG são comparados com os resultados obtidos pelo *Scatter Search* (SS) disponibilizados por Benavides et al. [4] e pela resolução do modelo matemático. A resolução do modelo matemático é realizado pelo solver IBM ILOG CPLEX sobre duas circunstancias: definindo uma da solução inicial (definida na seção 4.1) no modelo, apresentado nas tabelas como CPLEX ini, e a resolução do modelo sem definir uma solução inicial, denominado como CPLEX. Para ambos algoritmos, VNS-IG e TS-IG, foram realizadas 10 execuçoes para cada instância. Para o SS, Benavides et al. [4] realizaram 50 execuções do algoritmo para cada instância, ou seja, o SS tem vantagem na obtenção de melhores soluções.

Nas Tabelas 5 e 7 são apresentadas as médias dos valores do RPD encontradas pelos métodos utilizando as instâncias de Carlier. Os melhores valores encontrados do RPD estão destacados em negrito. Os resultados encontrados tanto pelos algoritmos baseados em meta-heurísticas quanto pelo CPLEX e CPLEX ini são similares. A variação de RPD entre os métodos são minímas, com exceção da instância car8I20 onde CPLEX e CPLEX ini conseguiram obter uma solução melhor que as soluções encontradas pelo VNS-IG, TS-IG e SS.

Para as instâncias de Carlier o CPLEX e CPLEX ini conseguiram provar a otimalidade para 19 instâncias, 16 a mais que Benavides et al. [4] e 6 a mais que Araujo et al. [27]. As 19 instâncias com solução ótima estão destacadas em negrito nas Tabelas 5 e 7 (primeira coluna). O único método a encontrar 100% das soluções ótimas foi o CPLEX ini, na sequência o CPLEX e SS encontraram 18 das 19 soluções ótimas (94%), o VNS-IG encontrou 15 dessas soluções ótimas (78%) e TS-IG encontrou apenas 14 soluções ótimas, o equivalente a 73% das soluções ótimas.

Nas tabelas 6 e 8 são apresentadas as médias dos valores do RPD encontradas pelos métodos utilizando as instâncias de Taillard. As 360 instâncias estão agrupadas em 36 grupos (com 10 instâncias), de acordo com o tamanho das instâncias (n x m), o intervalo utilizados para gerar os tempos de processamentos e a porcentagem de incompatibilidade entre trabalhadores e máquinas. Por exemplo, a instância ta025i10 é baseada na instancia 25 de Taillard com tempo intervalo de tempo de processamento variando de $[p_{ij}; 2p_{ij}]$ e 10% de incompatibilidade entre trabalhadores

e máquinas. O RPD nesse caso trata-se das médias das 10 instâncias pertencente a um grupo. Para esses grupos o TS-IG obteve a menor média de RPD para 18 grupos, o VNS-IG conseguiu encontrar os melhores valores de RPD para 16 grupos e o SS apenas para 2 grupos.

As instâncias de Taillard consideradas grandes, logo a resolução dessas instâncias de maneira exata é muito mais complexa devido ao seu extenso do espaço de busca. O CPLEX ini obteve um valor de RPD muito menor se comparado ao CPLEX, sendo melhor em 97% dos grupos, isso comprova a qualidade da solução inicial gerada a partir da heurística construtiva proposta. Além disso o CPLEX não consegue encontrar solução viável para 18 instâncias do problema após os 3600 segundos de execução.

As tabelas 9, 10, 11 e 12 da Seção Apêndice deste trabalho (Seção 9), apresentam os valores absolutos do *makespan* encontrados pelos os métodos apresentado neste trabalho.

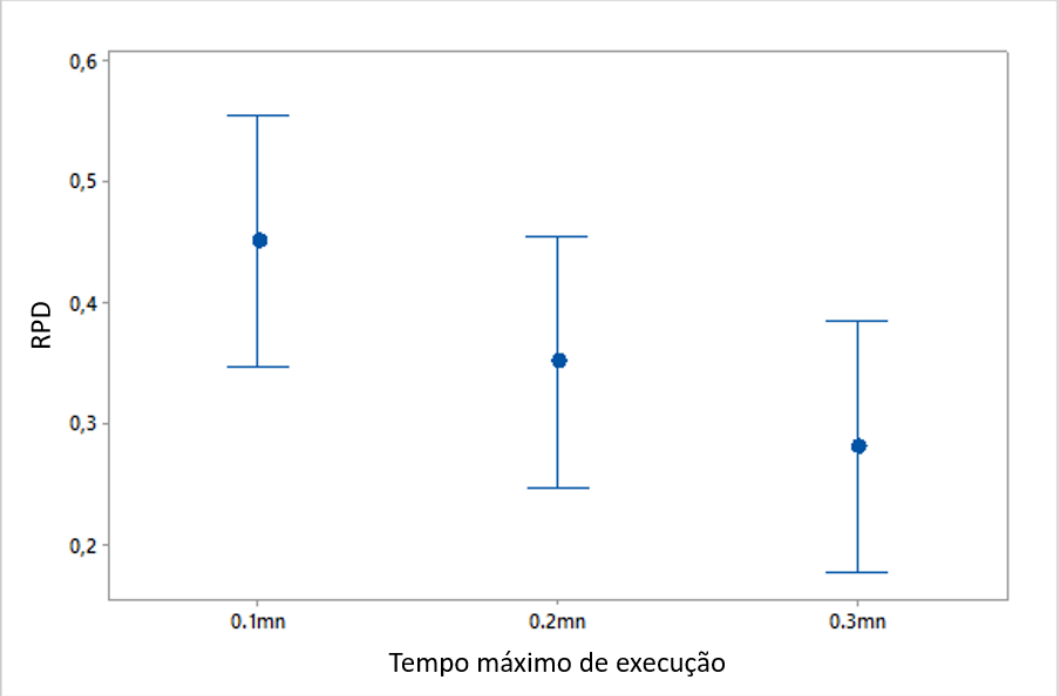


Figura 5: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração do tempo máximo de execução do VNS-IG

O gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95% para confrontar os algoritmos VNS-IG, TS-IG, SS, CPLEX ini e CPLEX utilizando as instâncias baseadas em Carlier é apresentado na Figura 7. A Figura 7 mostra que os intervalos das médias do RPD entre os métodos utilizados para resolução do problema se sobrepõem (com excessão do CPLEX ini, que obteve a pior média para esse grupo), logo os métodos testados para esse grupo de instancia não

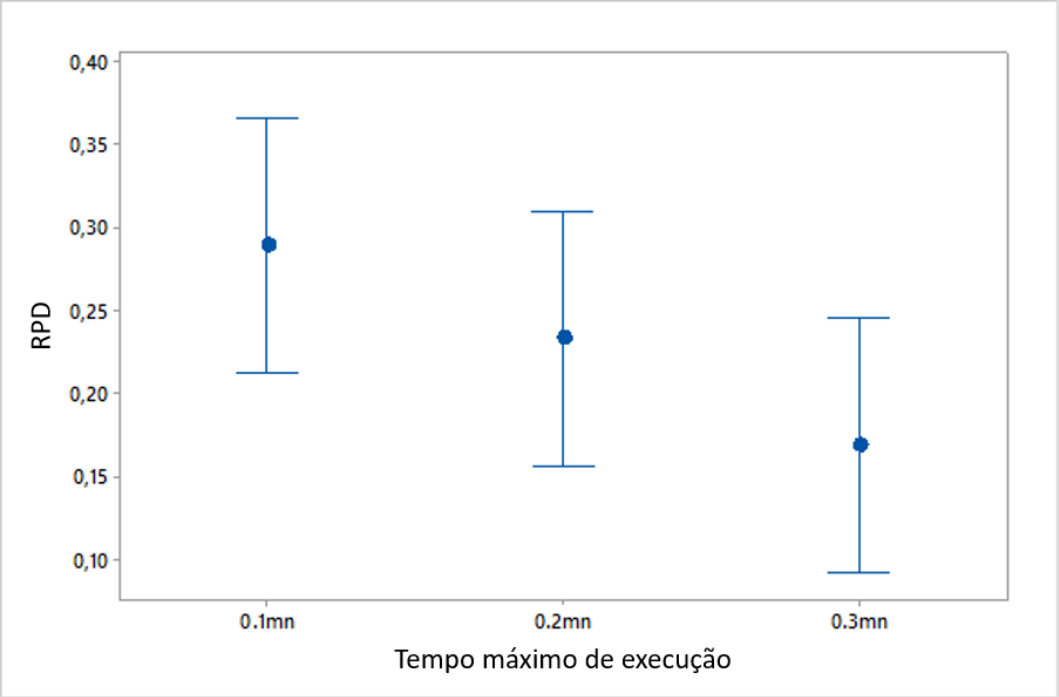
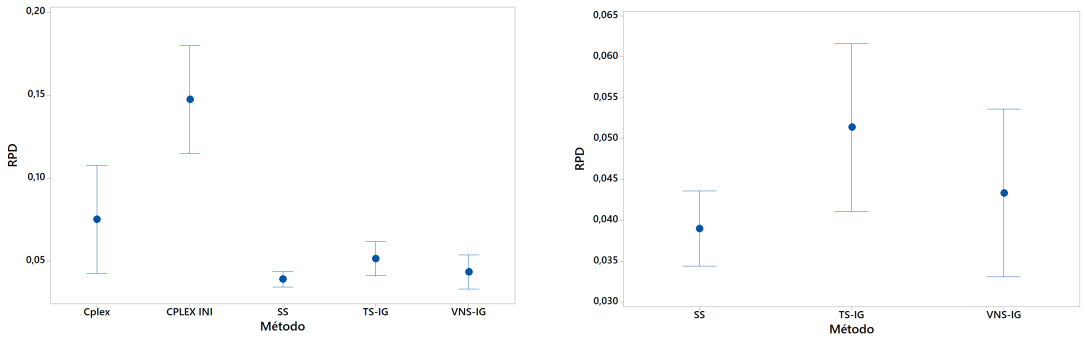


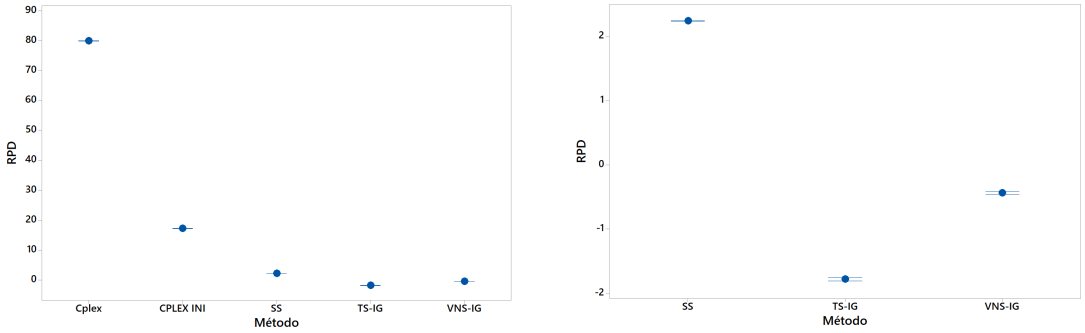
Figura 6: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração do tempo máximo de execução do TS-IG

apresentaram diferenças significativas, ambos apresentaram boas médias de RPD. Entretanto para as instancias baseadas em Taillard, foram observadas diferenças significativas entre os métodos de resolução apresentados, como mostra a Figura 8, ou seja, os resultados encontrados são significativamente diferente. As heurísticas propostas apresentam uma média de RPD inferior as encontradas pelos métodos SS, CPLEX e CPLEX ini. O algoritmo TS-IG é o algoritmo que obteve as melhores médias de RPD para esse problema. A discrepância de médias entre o CPLEX e CPLEX ini comprova a importância e qualidade da solução inicial gerada pelo algoritmo construtivo proposto. Durante os experimentos o TS-IG conseguiu encontrar 188 soluções com *makespan* menor que o UB, o VNS-IG encontrou 135 soluções e o SS apenas 3.



(a) Todos os métodos. (b) VNS-IG, TS-IG e SS.

Figura 7: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos algoritmos VNS-IG, TS-IG, SS e CPLEX para instâncias baseadas em Carlier.



(a) Todos os métodos. (b) VNS-IG, TS-IG e SS.

Figura 8: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos algoritmos VNS-IG, TS-IG, SS e CPLEX para instâncias baseadas em Taillard.

Tabela 6: Resultados das médias dos RPDs para os grupos de instâncias de Taillard com tempos de processamento em $[t; 2t]$ dos algoritmos comparados.

Grupo Instâncias	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
ta001i00-ta010i00	20	5	1727,70	1842,20	**0,08**	0,22	0,15	5,76	4,51
ta001i10-ta010i10	20	5	1731,40	1842,40	**0,04**	0,12	0,06	5,13	3,59
ta001i20-ta010i20	20	5	1741,00	1843,90	**0,03**	0,14	**0,03**	5,41	4,09
ta011i00-ta020i00	20	10	2086,30	2333,70	0,79	**0,51**	0,66	18,41	9,78
ta011i10-ta020i10	20	10	2091,00	2321,50	0,76	0,70	**0,69**	14,32	10,05
ta011i20-ta020i20	20	10	2103,40	9506,70	0,97	**0,69**	0,76	17,59	9,78
ta021i00-ta030i00	20	20	3058,70	3333,00	1,18	**-1,79**	0,99	28,79	8,27
ta021i10-ta030i10	20	20	3039,40	3380,60	2,07	**-0,92**	1,40	42,31	12,52
ta021i20-ta030i20	20	20	3037,60	17928,90	2,38	**-0,53**	1,25	32,88	13,54
ta031i00-ta040i00	50	5	3993,40	4183,10	**0,02**	0,06	0,35	17,90	4,68
ta031i10-ta040i10	50	5	3995,80	4130,40	**0,02**	0,07	0,37	20,45	6,12
ta031i20-ta040i20	50	5	4007,60	4159,00	**0,01**	0,08	0,35	18,42	8,28
ta041i00-ta050i00	50	10	4358,60	4629,30	-0,43	**-0,62**	1,73	78,05	6,21
ta041i10-ta050i10	50	10	4356,00	4631,90	0,06	**-0,25**	2,01	69,70	6,34
ta041i20-ta050i20	50	10	4358,40	100056,50	-0,02	**-0,18**	2,08	78,27	19,36
ta051i00-ta060i00	50	20	5567,60	5797,80	-0,68	**-3,21**	2,10	198,22	4,13
ta051i10-ta060i10	50	20	5563,10	5848,10	-0,27	**-3,05**	2,13	162,02	4,92
ta051i20-ta060i20	50	20	5565,20	53836,40	-0,04	**-2,84**	2,13	125,41	16,97

Tabela 5: Resultados das médias dos RPDs para as instâncias de Carlier com tempos de processamento em $[t; 2t]$ dos algoritmos comparados.

Instância	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
car1i00	11	5	9952	10569	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car1i10	11	5	9952	10569	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car1i20	11	5	9952	11307	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car2i00	13	4	10224	10892	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car2i10	13	4	10224	10892	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car2i20	13	4	10398	10892	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car3i00	12	5	10268	11956	**0,00**	**0,00**	**0,00**	1,26	1,26
car3i10	12	5	10359	10845	**0,00**	**0,00**	**0,00**	0,37	**0,00**
car3i20	12	5	10359	11557	**0,00**	**0,00**	**0,00**	0,37	**0,00**
car4i00	14	4	11613	12643	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car4i10	14	4	11846	12211	**0,00**	**0,00**	**0,00**	**0,00**	0,70
car4i20	14	4	11876	12211	**0,00**	**0,00**	**0,00**	0,13	**0,00**
car5i00	10	6	10589	11671	0,14	0,03	**0,00**	**0,00**	0,34
car5i10	10	6	10589	11817	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car5i20	10	6	10848	12136	**0,00**	**0,00**	**0,00**	**0,00**	0,76
car6i00	8	9	11044	12416	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car6i10	8	9	11044	12085	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car6i20	8	9	11118	12505	**0,00**	**0,00**	0,10	**0,00**	0,63
car7i00	7	7	8558	9438	0,20	0,26	**0,00**	**0,00**	**0,00**
car7i10	7	7	8558	9673	0,20	0,27	**0,00**	**0,00**	**0,00**
car7i20	7	7	8558	9105	0,20	0,28	**0,00**	**0,00**	**0,00**
car8i00	8	8	11533	13002	**0,00**	**0,00**	**0,00**	0,36	**0,00**
car8i10	8	8	11659	12185	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car8i20	8	8	11742	12462	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**

Tabela 8: Resultados das médias dos RPDs para os grupos de instâncias de Taillard com tempos de processamento em $[t; 5t]$ dos algoritmos comparados.

Grupo Instâncias	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
ta001I00-ta010I00	20	5	3296,60	3645,00	**0,02**	0,56	0,28	5,27	6,92
ta001I10-ta010I10	20	5	3313,20	3647,60	**0,01**	0,45	0,13	5,59	5,70
ta001I20-ta010I20	20	5	3343,70	3687,50	**0,07**	0,51	0,22	5,79	4,52
ta011I00-ta020I00	20	10	3917,20	4518,70	**0,17**	0,27	1,46	21,15	13,41
ta011I10-ta020I10	20	10	3944,30	4563,50	**-0,10**	0,57	1,38	20,37	14,55
ta011I20-ta020I20	20	10	3995,00	40330,10	**0,14**	0,73	1,50	20,40	12,57
ta021I00-ta030I00	20	20	5742,80	6468,20	0,40	**-3,56**	2,48	50,75	11,99
ta021I10-ta030I10	20	20	5756,70	6469,60	1,07	**-2,81**	2,39	41,91	17,85
ta021I20-ta030I20	20	20	5796,20	24825,50	1,57	**-2,60**	1,75	33,54	18,51
ta031I00-ta040I00	50	5	7853,40	8299,00	**0,01**	0,04	0,52	22,09	5,67
ta031I10-ta040I10	50	5	7860,20	8250,20	**0,02**	0,03	0,47	22,30	7,83
ta031I20-ta040I20	50	5	7875,90	8334,60	**-0,02**	0,17	0,64	20,78	10,08
ta041I00-ta050I00	50	10	8445,10	9060,60	**-1,40**	**-1,40**	4,85	88,23	7,29
ta041I10-ta050I10	50	10	8449,30	9069,40	**-1,16**	0,84	5,00	95,37	7,22
ta041I20-ta050I20	50	10	8488,10	246838,70	**-1,18**	-0,83	4,75	94,34	33,55
ta051I00-ta060I00	50	20	11114,50	11353,60	-2,94	**-6,43**	4,23	205,85	2,15
ta051I10-ta060I10	50	20	11151,40	131380,70	-2,39	**-6,13**	4,00	218,21	75,86
ta051I20-ta060I20	50	20	11152,50	370547,20	-2,21	**-5,69**	4,12	281,62	60,71

Tabela 7: Resultados das médias dos RPDs para as instâncias de Carlier com tempos de processamento em $[t; 5t]$ dos algoritmos comparados.

Instância	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
car1I00	11	5	19508	23024	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car1I10	11	5	19831	23024	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car1I20	11	5	19831	23024	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car2I00	13	4	19876	21264	**0,00**	**0,00**	**0,00**	**0,00**	0,01
car2I10	13	4	19876	21264	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car2I20	13	4	19876	21264	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car3I00	12	5	19498	21230	**0,00**	**0,00**	0,10	0,29	0,29
car3I10	12	5	19498	21230	**0,00**	**0,00**	0,08	0,29	0,29
car3I20	12	5	19498	21739	**0,00**	**0,00**	0,03	0,29	0,29
car4I00	14	4	20381	22523	**0,00**	**0,00**	**0,00**	0,20	0,20
car4I10	14	4	22270	24385	**0,00**	**0,00**	**0,00**	**0,00**	1,00
car4I20	14	4	22270	22884	**0,00**	**0,00**	**0,00**	**0,00**	1,03
car5I00	10	6	18693	19666	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car5I10	10	6	18693	19666	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car5I20	10	6	19468	22388	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car6I00	8	9	20070	24481	**0,00**	**0,00**	**0,00**	0,34	**0,00**
car6I10	8	9	20070	25036	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car6I20	8	9	20070	22581	**0,00**	0,24	**0,00**	**0,00**	**0,00**
car7I00	7	7	16423	16962	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car7I10	7	7	17067	20165	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car7I20	7	7	17257	19012	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car8I00	8	8	19159	21444	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car8I10	8	8	19159	20236	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
car8I20	8	8	19791	21628	1,25	1,31	1,17	**0,00**	**0,00**

7 Conclusão

Nesse trabalho é proposto uma heurística construtiva e dois algoritmos híbridos para a resolução do problema de FSNPTH de maneira aproximada em tempo computacional razoável.

Os resultados obtidos pelos algoritmos são comparados com os resultados encontrados pelo algoritmo *Scatter Search* (SS) proposto por Benavides et al.[4] e a resolução do modelo matemático pelo solver IBM ILOG CPLEX sobre duas circunstâncias, a partir de uma solução inicial e sem a solução inicial. Os testes computacionais apresentados comprovam a eficiência do TS-IG e VNS-IG em relação ao SS e resolução do modelo matemático. Além disso o algoritmo TS-IG se mostrou melhor que VNS-IG para esse problema. Os resultados obtidos foram validados através de testes estatísticos. Acredita-se que os resultados apresentados são os melhores resultados que existem até o momento na literatura para o problema FSNPTH.

Como trabalhos futuros pretende-se desenvolver novos algoritmos fundamentados em outras meta-heurísticas, além de melhorar o desempenho dos algoritmos propostos.

8 Agradecimentos

Os autores agradecem à CAPES, CNPq, FAPEMIG e a Divisão de Apoio ao Desenvolvimento Científico e Tecnológico da Universidade Federal de Viçosa.

9 Apêndice

As Tabelas 9, 10, 11 e 12 complementam os resultados obtidos na Seção 6 e apresentam os valores médios do *makespan* obtidos durante os testes computacionais definidos na seção 5 pelos métodos comparados neste trabalho.

Tabela 9: Resultados das médias dos *makespan* para as instâncias de Carlier com tempos de processamento em $[t; 2t]$ dos algoritmos comparados.

Instância	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
car1i00	11	5	9952	10569	**9952**	**9952**	**9952**	**9952**	**9952**
car1i10	11	5	9952	10569	**9952**	**9952**	**9952**	**9952**	**9952**
car1i20	11	5	9952	11307	**9952**	**9952**	**9952**	**9952**	**9952**
car2i00	13	4	10224	10892	**10224**	**10224**	**10224**	**10224**	**10224**
car2i10	13	4	10224	10892	**10224**	**10224**	**10224**	**10224**	**10224**
car2i20	13	4	10398	10892	**10398**	**10398**	**10398**	**10398**	**10398**
car3i00	12	5	10268	11956	**10268**	**10268**	**10268**	10397	10397
car3i10	12	5	10359	10845	**10359**	**10359**	**10359**	10397	**10359**
car3i20	12	5	10359	11557	**10359**	**10359**	**10359**	10397	**10359**
car4i00	14	4	11613	12643	**11613**	**11613**	**11613**	**11613**	**11613**
car4i10	14	4	11846	12211	**11846**	**11846**	**11846**	**11846**	11929
car4i20	14	4	11876	12211	**11876**	**11876**	**11876**	11892	**11876**
car5i00	10	6	10589	11671	10603,4	10592,6	**10589**	**10589**	10625
car5i10	10	6	10589	11817	**10589**	**10589**	**10589**	**10589**	**10589**
car5i20	10	6	10848	12136	**10848**	**10848**	**10848**	**10848**	10930
car6i00	8	9	11044	12416	**11044**	**11044**	**11044**	**11044**	**11044**
car6i10	8	9	11044	12085	**11044**	**11044**	**11044**	**11044**	**11044**
car6i20	8	9	11118	12505	**11118**	**11118**	11129,2	**11118**	**11188**
car7i00	7	7	8558	9438	8575	8580	**8558**	**8558**	**8558**
car7i10	7	7	8558	9673	8575	8580,7	**8558**	**8558**	**8558**
car7i20	7	7	8558	9105	8575	8580,6	**8558**	**8558**	
car8i00	8	8	11533	13002	**11533**	**11533**	**11533**	11575	**11533**
car8i10	8	8	11659	12185	**11659**	**11659**	**11659**	**11659**	**11659**
car8i20	8	8	11742	12462	**11742**	**11742**	**11742**	**11742**	**11742**

Tabela 10: Resultados das médias dos *makespan* para as instâncias de Carlier com tempos de processamento em $[t; 5t]$ dos algoritmos comparados.

Instância	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
car1I00	11	5	19508	23024	**19508**	**19508**	**19508**	**19508**	**19508**
car1I10	11	5	19831	23024	**19831**	**19831**	**19831**	**19831**	**19831**
car1I20	11	5	19831	23024	**19831**	**19831**	**19831**	**19831**	**19831**
car2I00	13	4	19876	21264	**19876**	**19876**	**19876**	**19876**	19877
car2I10	13	4	19876	21264	**19876**	**19876**	**19876**	**19876**	**19876**
car2I20	13	4	19876	21264	**19876**	**19876**	**19876**	**19876**	**19876**
car3I00	12	5	19498	21230	**19498**	**19498**	19517	19554	19554
car3I10	12	5	19498	21230	**19498**	**19498**	19513	19554	19554
car3I20	12	5	19498	21739	**19498**	**19498**	19504	19554	19554
car4I00	14	4	20381	22523	**20381**	**20381**	**20381**	20422	20422
car4I10	14	4	22270	24385	**22270**	**22270**	**22270**	**22270**	22493
car4I20	14	4	22270	22884	**22270**	**22270**	**22270**	**22270**	22499
car5I00	10	6	18693	19666	**18693**	**18693**	**18693**	**18693**	**18693**
car5I10	10	6	18693	19666	**18693**	**18693**	**18693**	**18693**	**18693**
car5I20	10	6	19468	22388	**19468**	**19468**	**19468**	**19468**	**19468**
car6I00	8	9	20070	24481	**20070**	**20070**	**20070**	20139	**20070**
car6I10	8	9	20070	25036	**20070**	**20070**	**20070**	**20070**	**20070**
car6I20	8	9	20070	22581	**20070**	20090,8	**20070**	**20070**	**20070**
car7I00	7	7	16423	16962	**16423**	**16423**	**16423**	**16423**	**16423**
car7I10	7	7	17067	20165	**17067**	**17067**	**17067**	**17067**	**17067**
car7I20	7	7	17257	19012	**17257**	**17257**	**17257**	**17257**	**17257**
car8I00	8	8	19159	21444	**19159**	**19159**	**19159**	**19159**	**19159**
car8I10	8	8	19159	20236	**19159**	**19159**	**19159**	**19159**	**19159**
car8I20	8	8	19791	21628	20038	20049,8	20022	**19791**	**19791**

Tabela 11: Resultados das médias dos *makespan* para os grupos de instâncias de Taillard com tempos de processamento em $[t; 2t]$ dos algoritmos comparados.

Grupo Instâncias	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
ta001i00-ta010i00	20	5	1727,70	1842,20	**1729,09**	1731,45	1730,28	1827,20	1805,60
ta001i10-ta010i10	20	5	1731,40	1842,40	**1732,12**	1733,52	1732,36	1820,20	1793,60
ta001i20-ta010i20	20	5	1741,00	1843,90	1741,57	1743,52	**1741,56**	1835,20	1812,20
ta011i00-ta020i00	20	10	2086,30	2333,70	2102,80	**2096,79**	2099,99	2470,30	2290,30
ta011i10-ta020i10	20	10	2091,00	2321,50	2106,82	2105,64	**2105,33**	2390,50	2301,10
ta011i20-ta020i20	20	10	2103,40	9506,70	2123,76	**2117,12**	2119,29	2472,80	2309,20
ta021i00-ta030i00	20	20	3058,70	3333,00	3094,67	**3003,36**	3088,95	3939,40	3311,80
ta021i10-ta030i10	20	20	3039,40	3380,60	3102,42	**3011,34**	3082,02	4325,40	3419,90
ta021i20-ta030i20	20	20	3037,60	17928,90	3110,03	**3021,39**	3075,52	4036,50	3449,00
ta031i00-ta040i00	50	5	3993,40	4183,10	**3994,37**	3995,83	4007,38	4708,30	4180,40
ta031i10-ta040i10	50	5	3995,80	4130,40	**3996,50**	3998,80	4010,49	4813,00	4240,30
ta031i20-ta040i20	50	5	4007,60	4159,00	**4008,10**	4010,62	4021,58	4745,70	4339,50
ta041i00-ta050i00	50	10	4358,60	4629,30	4339,67	**4331,58**	4434,13	7760,50	4629,30
ta041i10-ta050i10	50	10	4356,00	4631,90	4358,42	**4344,83**	4443,35	7392,20	4632,10
ta041i20-ta050i20	50	10	4358,40	100056,50	4357,59	**4350,65**	4449,17	7769,60	5202,10
ta051i00-ta060i00	50	20	5567,60	5797,80	5529,91	**5388,63**	5684,66	16603,75	5797,80
ta051i10-ta060i10	50	20	5563,10	5848,10	5548,24	**5393,51**	5681,67	14576,25	5836,90
ta051i20-ta060i20	50	20	5565,20	53836,40	5563,01	**5407,00**	5683,90	12544,40	6509,80

Tabela 12: Resultados das médias do *makespan* para os grupos de instâncias de Taillard com tempos de processamento em $[t; 5t]$ dos algoritmos comparados.

Grupo Instâncias	Tamanho		ub	SOL ini	RPD				
	n	m			VNS-IG	TS-IG	SS	CPLEX	CPLEX ini
ta001I00-ta010I00	20	5	3296,60	3645,00	**3297,10**	3314,63	3305,83	3470,20	3524,60
ta001I10-ta010I10	20	5	3313,20	3647,60	**3313,69**	3327,79	3317,66	3498,50	3502,20
ta001I20-ta010I20	20	5	3343,70	3687,50	**3346,05**	3357,20	3351,05	3537,40	3494,90
ta011I00-ta020I00	20	10	3917,20	4518,70	**3923,84**	3927,33	3974,38	4745,70	4442,50
ta011I10-ta020I10	20	10	3944,30	4563,50	**3940,19**	3966,22	3998,77	4747,60	4518,20
ta011I20-ta020I20	20	10	3995,00	40330,10	**4000,74**	4023,68	4054,73	4809,90	4497,00
ta021I00-ta030I00	20	20	5742,80	6468,20	5765,83	**5537,46**	5885,24	8657,50	6431,50
ta021I10-ta030I10	20	20	5756,70	6469,60	5818,50	**5593,21**	5894,40	8169,20	6784,30
ta021I20-ta030I20	20	20	5796,20	24825,50	5887,39	**5643,40**	5897,45	7740,20	6869,30
ta031I00-ta040I00	50	5	7853,40	8299,00	**7854,35**	7856,72	7893,88	9588,60	8298,50
ta031I10-ta040I10	50	5	7860,20	8250,20	**7861,56**	7862,29	7896,79	9612,70	8475,90
ta031I20-ta040I20	50	5	7875,90	8334,60	**7874,60**	7889,08	7926,62	9512,40	8669,80
ta041I00-ta050I00	50	10	8445,10	9060,60	**8326,69**	8327,25	8854,52	15895,80	9060,60
ta041I10-ta050I10	50	10	8449,30	9069,40	**8350,87**	8378,31	8872,15	16507,80	9059,00
ta041I20-ta050I20	50	10	8488,10	246838,70	**8387,70**	8418,11	8891,27	16495,40	11336,20
ta051I00-ta060I00	50	20	11114,50	11353,60	10788,24	**10400,24**	11584,49	33993,70	11353,60
ta051I10-ta060I10	50	20	11151,40	131380,70	10884,33	**10468,14**	11597,19	35484,67	19610,50
ta051I20-ta060I20	50	20	11152,50	370547,20	10906,33	**10517,33**	11612,22	42559,67	17922,70

Referências

[1] P. Brucker, Scheduling Algorithms, 5th Edition, Springer-Verlag Berlin Heidelberg, 2007.

[2] S. M. Johnson, Optimal two-and three-stage production schedules with setup times included, Naval Research Logistics (NRL) 1 (1) (1954) 61–68.

[3] M. R. Garey, D. S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of operations research 1 (2) (1976) 117–129.

[4] A. J. Benavides, M. Ritt, C. Miralles, Flow shop scheduling with heterogeneous workers, European Journal of Operational Research 237 (2) (2014) 713–720.

[5] A. A. Chaves, Uma meta-heurística híbrida com busca por agrupamentos aplicada a problemas de otimização combinatória, Instituto Nacional de Pesquisas Espaciais.

[6] Brasil, Secretaria de direitos humanos presidência da república, `http://www.sdh.gov.br/assuntos/pessoa-com-deficiencia/dados-estatisticos/pesquisas-demografica` acessado: 2016-04-10 (2010).

[7] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & operations research 24 (11) (1997) 1097–1100.

[8] F. Glover, Tabu search-part i, ORSA Journal on computing 1 (3) (1989) 190–206.

[9] T. G. Crainic, M. Gendreau, P. Soriano, M. Toulouse, A tabu search procedure for multicommodity location/allocation with balancing requirements, Annals of Operations research 41 (4) (1993) 359–383.

[10] L. Demir, S. Tunalı, D. T. Eliiyi, A. Løkketangen, Two approaches for solving the buffer allocation problem in unreliable production lines, Computers & Operations Research 40 (10) (2013) 2556–2563.

[11] Y. Ren, A. Awasthi, Investigating metaheuristics applications for capacitated location allocation problem on logistics networks, in: Chaos Modeling and Control Systems Design, Springer, 2015, pp. 213–238.

[12] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (3) (2007) 2033–2049.

[13] A. J. Benavides, M. Ritt, Iterated local search heuristics for minimizing total completion time in permutation and non-permutation flow shops., in: ICAPS, 2015, pp. 34–41.
[14] M. A. F. Belo Filho, Programação de produção e dimensionamento de lotes para flowshop, Ph.D. thesis, Universidade de São Paulo (2010).
[15] B. L. Maccarthy, J. Liu, Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling, The International Journal of Production Research 31 (1) (1993) 59–79.
[16] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2) (2005) 479–494.
[17] C. N. Potts, V. A. Strusevich, Fifty years of scheduling: a survey of milestones, Journal of the Operational Research Society (2009) S41–S68.
[18] J. N. Gupta, E. F. Stafford, Flowshop scheduling research after five decades, European Journal of Operational Research 169 (3) (2006) 699–711.
[19] M. Nawaz, E. E. Enscore, I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11 (1) (1983) 91–95.
[20] M. Tandon, P. Cummings, M. LeVan, Flowshop sequencing with non-permutation schedules, Computers & chemical engineering 15 (8) (1991) 601–607.
[21] P. Brucker, S. Heitmann, J. Hurink, Flow-shop problems with intermediate buffers, OR Spectrum 25 (4) (2003) 549–574.
[22] C. Liao, L. Liao, C. Tseng, A performance evaluation of permutation vs. non-permutation schedules in a flowshop, International Journal of Production Research 44 (20) (2006) 4297–4309.
[23] B. Yagmahan, M. M. Yenisey, A multi-objective ant colony system algorithm for flow shop scheduling problem, Expert Systems with Applications 37 (2) (2010) 1361–1368.
[24] A. Rossi, M. Lanzetta, Scheduling flow lines with buffers by ant colony digraph, Expert Systems with Applications 40 (9) (2013) 3328–3340.
[25] K.-C. Ying, Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic, The International Journal of Advanced Manufacturing Technology 38 (3-4) (2008) 348–354.
[26] A. J. Benavides, M. Ritt, Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops, Computers & Operations Research 66 (2016) 160–169.
[27] J. E. C. . S. A. G. Araujo, M. F.; Arroyo, Método proximity search para a resolução do problema de flow shop scheduling não permutacional com trabalhadores heterogêneos (fsnpth)., Anais do XLVIII SBPO Simpósio Brasileiro de Pesquisa Operacional.
[28] G. C. Carniel, A. J. Benavides, M. Ritt, C. Miralles, Including workers with disabilities in flow shop scheduling, in: Automation Science and Engineering (CASE), 2015 IEEE International Conference on, IEEE, 2015, pp. 985–991.
[29] M. C. d. O. Moreira, Problema de balanceamento de linhas de produção e integração de trabalhadores, Ph.D. thesis, Universidade de São Paulo (2016).
[30] R. Mencía, M. R. Sierra, C. Mencía, R. Varela, Genetic algorithms for the scheduling problem with arbitrary precedence relations and skilled operators, Integrated Computer-Aided Engineering 23 (3) (2016) 269–285.
[31] A. J. Benavides, M. Ritt, C. Miralles, Heterogeneous workforce in job shop scheduling.
[32] M. R. Bonyadi, Z. Michalewicz, L. Barone, The travelling thief problem: the first step in the transition from theoretical problems to realistic problems, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, 2013, pp. 1037–1044.
[33] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job shop problem, Management science 42 (6) (1996) 797–813.
[34] J. Carlier, Ordonnancements a contraintes disjonctives, Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle 12 (4) (1978) 333–350.

[35] E. Taillard, Benchmarks for basic scheduling problems, european journal of operational research 64 (2) (1993) 278–285.
[36] J. H. Zar, Biostatistical analysis. 4th, New Jersey, USA (1999) 929.
[37] I. Corporation, Intel® core^{TM} i7-6700k processor specifications, accessed: 10 Jan. 2017 (Jan. 2017).
URL https://ark.intel.com/products/88195/Intel-Core-i7-6700K-Processor-8M-Cache-up-to-4_20-GHz
[38] AMD, Amd opteron 4200 series processor quick reference guide, accessed: 10 Jan. 2017 (Jun. 2012).
URL https://www.amd.com/Documents/Opteron$_4000_QRG.pdf$
[39] CPUBoss, Amd opteron 4230 he, accessed: 10 Jan. 2017 (Jun. 2012).
URL http://cpuboss.com/cpu/AMD-Opteron-4230-HE
[40] CPUBoss, Intel core i7 6700k, accessed: 10 Jan. 2017 (Jul. 2015).
URL http://cpuboss.com/cpu/Intel-Core-i7-6700K
[41] CNET, Amd opteron 4230 he / 2.9 ghz processor, accessed: 10 Jan. 2017 (Jan. 2017).
URL https://www.cnet.com/products/amd-opteron-4230-he-2-9-ghz-processor/specs/
[42] CNET, Intel core i7 6700k / 4 ghz processor, accessed: 10 Jan. 2017 (Jan. 2017).
URL https://www.cnet.com/products/intel-core-i7-6700k-4-ghz-processor/specs/
[43] CPU-World, Amd opteron 4230 he specifications, accessed: 10 Jan. 2017 (Dec. 2016).
URL http://www.cpu-world.com/CPUs/Bulldozer/AMD-Opteron%204230%20HE%20(OS4230OFU6KGU).html
[44] CPU-World, Intel core i7-6700k specifications, accessed: 10 Jan. 2017 (Jan. 2017).
URL http://www.cpu-world.com/CPUs/Core$_i7/Intel-Core\%20i7-6700K.html$
[45] CPUBoss, Intel core i7 6700k vs amd opteron 4230 he", month = jul, year = 2015, url = http://cpuboss.com/cpus/Intel-Core-i7-6700K-vs-AMD-Opteron-4230-HE, note = Accessed: 10 Jan. 2017.

Capítulo 4

Conclusões

Neste trabalho foi abordado o problema de *flow shop scheduling* não permutacional com trabalhadores heterogêneos (FSNPTH). O objetivo consiste em alocar trabalhadores heterogêneos em máquinas, nas quais a heterogeneidade dos trabalhadores é definida em relação ao tempo que cada um gasta para operar uma determinada máquina, e a alocação das tarefas em máquinas em séries de forma que minimize o *makespan* de um sistema de produção *flow shop*.

O problema abordado possui grande importância no cenário mundial, uma vez que, as taxas de desemprego sofridas pelos deficientes é um problema preocupante em diversos países. Além disso, indústrias e empresas utilizam mão de obra de pessoas com deficiência e o sistema de produção *flow shop* pode alocar esses trabalhadores de maneira que suas deficiências e/ou falta de experiência e qualificação influenciem o mínimo possível no processo de produção. Outro ponto importante é que o problema de FSNPTH pertence à classe de problemas NP-difícil (Garey et al., 1976) e também é classificado como multicomponente, além do mais é um problema relativamente novo onde suas características são pouco abordadas na literatura.

Benavides et al. (2014) propõem um modelo de programação linear inteira mista 0-1 (PLIM) para resolução do problema de FSNPTH, entretanto, os resultados obtidos comprovam que se trata de um problema de difícil resolução de maneira exata até mesmo para instâncias pequenas. Com o objetivo de resolver o problema de maneira exata, foi aplicado o método *Proximity Search* (PS) proposto

por Fischetti & Monaci (2014). De acordo com os experimentos computacionais realizados, todas as versões propostas do PS obtiveram desempenho superior a resolução do modelo matemático pelo solver IBM ILOG CPLEX (IBM, 2011) em relação a qualidade das soluções encontradas e o tempo gasto para obter as soluções ótimas. Além disso, o PS conseguiu encontrar soluções para todas as instâncias do problema.

Com o objetivo de obter soluções de boa qualidade e em pouco tempo computacional, neste trabalho foram propostos dois algoritmos híbridos, o VNS-IG, que combinam as meta-heurísticas *Variable Neighborhood Search* (VNS) e *Iterated Greedy* (IG), e o TS-IG, que combina as meta-heurísticas Busca Tabu (TS, do inglês *Tabu Search*) e IG. Todos os parâmetros utilizados pelos algoritmos foram calibrados e validados através de testes estatísticos. De acordo com os experimentos realizados a qualidade das soluções encontradas pelos algoritmos híbridos é superior as encontradas pelo *Scatter Search* (SS) (Benavides et al., 2014), sendo o TS-IG o melhor nesse quesito. Além disso, ambos os algoritmos propostos superam o SS no quesito tempo de execução. Enquanto o SS fixa o tempo máximo de execução em 600 segundos para todas as instâncias, nos algoritmos VNS-IG e TS-IG, o tempo máximo de execução é definido de acordo com o tamanho da instância, sendo, 100 segundos o tempo máximo para a maior instancia do problema. Por fim, o algoritmo TS-IG encontrou soluções com o valor do *makespan* inferior ao do melhor valor conhecido para 188 instâncias e o VNS-IG encontrou 135 dessas soluções.

Referências Bibliográficas

AACD (2001). Associação de assistência à criança deficiente (programa trabalho eficiente). Disponível em http://www.aacd.org.br/. Acessado em 01/2017.

Antunes, R. (2004). A dialética do trabalho: escritos de marx e engels. *São Paulo: Expressão Popular*.

Barros, R. P. d.; Camargo, J. M. & Mendonça, R. (1997). A estrutura do desemprego no brasil.

Benavides, A. J. & Ritt, M. (2016). Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops. *Computers & Operations Research*, 66:160--169.

Benavides, A. J.; Ritt, M. & Miralles, C. (2014). Flow shop scheduling with heterogeneous workers. *European Journal of Operational Research*, 237(2):713--720.

Bonyadi, M. R.; Michalewicz, Z. & Barone, L. (2013). The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. Em *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pp. 1037--1044. IEEE.

Brasil (1999). Decreto nº 3.298, de 20 de dezembro de 1999. regulamenta a lei nº 7.853, de 24 de outubro de 1989, dispõe sobre a política nacional para a integração da pessoa portadora de deficiência, consolida as normas de proteção, e dá outras providências. *Diário Oficial da União*.

Brasil (2015). Lei brasileira de inclusão da pessoa com deficiência (estatuto da pessoa com deficiência). *Diário Oficial da União*.

Brucker, P. (2007). *Scheduling Algorithms*. Springer-Verlag Berlin Heidelberg, fifth edição.

Chaves, A. A. (2009). Uma meta-heurística híbrida com busca por agrupamentos aplicada a problemas de otimização combinatória. *Instituto Nacional de Pesquisas Espaciais*.

Costa, V. d. (2001). *A formação na perspectiva da teoria crítica da sociedade: as experiências dos trabalhadores deficientes visuais do serviço federal de processamento de dados*. PhD thesis, Tese (Doutorado em Educação)-Programa de Educação, História e Filosofia da Educação. São Paulo: Pontifícia Universidade Católica de São Paulo, 2001b.[Links].

CVI-Rio (1982). Centro de vida independente do rio de janeiro - cvi rio. Disponível em http://www.cvi-rio.org.br/. Acessado em 01/2017.

Diniz, D. (2007). O que é deficiência.

Fischetti, M. & Monaci, M. (2014). Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709--731.

Garey, M. R.; Johnson, D. S. & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117--129.

Giordano, B. W. (2000). *(D) eficiência e trabalho: analisando suas representações*, volume 141. Annablume.

Glover, F. (1989). Tabu search-part i. *ORSA Journal on computing*, 1(3):190--206.

Hillier, F. S. & Lieberman, G. J. (2013). *Introdução à pesquisa operacional*. McGraw Hill Brasil.

IBGE (2010). Características gerais da população, religião e pessoas com deficiência. *Rio de Janeiro: Instituto Brasileiro de Geografia e Estatística*.

IBGE (2016). Indicadores conjunturais em 2016. Disponível em http://www.ibge.gov.br/home/estatistica/pesquisas/indicadores.php. Acessado em 01/2017.

IBM (2011). Ibm - mathematical programming: Linear programming, mixed-integer programming and quadratic programming - ibm ilog cplex optimizer - software. Disponível em http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/. Acessado em 01/2017.

IECONOMICS (2017). Taxa de desemprego - lista de países. Disponível em http://pt.tradingeconomics.com/country-list/unemployment-rate. Acessado em 01/2017.

Mladenović, N. & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11):1097--1100.

Paim, P. (2015). Estatuto da pessoa com deficiência - lei brasileira de inclusão.

Reinert, J. N. (2001). Desemprego: causas, conseqüências e possíveis soluções. *Revista de Ciências da Administração*, 3(5):45--48.