SAULO CUNHA CAMPOS

APLICAÇÃO DE METAHEURÍSTICAS PARA O PROBLEMA DE PROGRAMAÇÃO DA PRODUÇÃO EM UM AMBIENTE ASSEMBLY FLOWSHOP COM TRÊS ESTÁGIOS E TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

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.*

VIÇOSA
MINAS GERAIS – BRASIL
2014

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

T
C198a
2015

Campos, Saulo Cunha, 1984-
Aplicação de metaheurísticas para o problema de programação da produção em um ambiente assembly flowshop com três estágios e tempos de preparação dependentes da sequência / Saulo Cunha Campos. – Viçosa, MG, 2015.
xv, 123f. : il. (algumas color.) ; 29 cm.

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

1. Assembler (Linguagem de programação de computador). 2. Algorítmicos. 3. Metaheurísticas. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22. ed. 005.13

SAULO CUNHA CAMPOS

APLICAÇÃO DE METAHEURÍSTICAS PARA O PROBLEMA DE PROGRAMAÇÃO DA PRODUÇÃO EM UM AMBIENTE ASSEMBLY FLOWSHOP COM TRÊS ESTÁGIOS E TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

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: 25 de fevereiro de 2015.

André Gustavo dos Santos

Heleno do Nascimento Santos

José Elias Claudio Arroyo
Orientador

Dedico este trabalho aos meus pais, Omar e Célia, e a minha esposa Natália.

A vocês, todo o meu amor, sempre.

AGRADECIMENTOS

Saber reconhecer os benefícios recebidos de alguém é algo nobre, por isso não posso perder a oportunidade de deixar registrado meus agradecimentos sinceros à todos aqueles que me ajudaram nessa caminhada. Em mim há um sentimento profundo de valorização e admiração por todos vocês. Saibam que vocês possibilitaram a realização de um sonho. Desejo que a vida, de alguma forma, possa retribuí-los em múltiplas porções o que a mim foi feito.

Primeiramente, agradeço aos meus pais, Omar e Célia. Muito obrigado pelo amor incondicional a mim dedicado, por todas as oportunidades que me concederam e por todo esforço feito para me educar. Eu sei que não foi fácil, por isso, mais uma vez, obrigado.

Agradeço a minha amada esposa Natália. Sei que foi difícil para você também. Foi um tempo de muita renuncia e compreensão. Na verdade, não tenho nem palavras para expressar o quanto lhe sou grato. Foram tantas revisões de textos, tantos favores feitos e inúmeras tarefas realizadas… Além disso, sempre pude contar com seu amor e carinho para me fortalecer. Muito obrigado! Sem você não teria conseguido chegar até aqui.

Agradeço a FAGOC como um todo. Ela vem transformando a minha vida desde 2004. Eu sou prova que a FAGOC verdadeiramente realiza sonhos.

Meu agradecimento especial ao amigo Marcelo Daibert. Você é um exemplo de ser humano e profissional. Saiba que você contribuiu muito e me motivou para eu estar aqui hoje. Somente por isso já lhe sou extremamente grato, mas além disso, agradeço também pela sua amizade, companheirismo e o apoio sempre presente.

Agradeço ao Eraldo, que foi quem acreditou no meu potencial pela primeira vez e abriu as portas da FAGOC para mim. Agradeço ao Clayton Fraga, que me deu a oportunidade de experimentar esse caminho maravilhoso da docência. Agradeço ao Marcelo Andrade, que além de me ensinar algoritmos, permitiu a flexibilização do meu horário de trabalho, possibilitando a conciliação com as minhas atividades acadêmicas.

Agradeço ao meu orientador José Elias Arroyo, por todo conhecimento a mim transmitido, pelos diversos artigos escritos e publicados e por todo o aprendizado. Agradeço a todos os professores do DPI. Em especial, a professora Luciana, pois foi na sua disciplina de meta-heurística que o caminho para este trabalho foi aberto.

Agradeço a todos os colegas de mestrado: Angélica, Marques, Paola, Renan, William, Ítalo, Pipico, Matheus, Rafhael, Leo, dentres outros. Um agradecimento especial aos amigos Vinícius e Harlem. Foram tantos experimentos, análises estatísticas, artigos e apresentações…

Agradeço a toda a equipe da TI da FAGOC. William, Pedro, Natália, Matheus, Felipe, Ézio… Vocês formam uma excelente equipe e isso facilita muito o meu trabalho. Saibam que indiretamente vocês me ajudaram muito a conseguir chegar até aqui.

Agradeço também ao meu irmão Sandro por sua amizade e companheirismo, mesmo estando longe.

Por fim, agradeço a todos os meus alunos. É por vocês que percorri este longo caminho. Para poder lhes ofertar melhores aulas, melhores orientações e uma qualidade maior de ensino. Muito obrigado por me motivarem a chegar até aqui.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

CAMPOS, Saulo Cunha, M.Sc., Universidade Federal de Viçosa, fevereiro de 2015. **Aplicação de metaheurísticas para o problema de programação da produção em ambiente Assembly Flowshop com três estágios e tempos de setup dependentes.** Orientador: José Elias Claúdio Arroyo.

Este trabalho aborda o problema *Assembly flowshop* de três estágios, onde existem *m* máquinas paralelas no primeiro estágio, uma máquina de transporte no segundo estágio e uma máquina de montagem no terceiro estágio. No primeiro estágio, diferentes partes do produto são fabricadas de forma independente nas máquinas paralelas. No segundo estágio, as peças fabricadas são coletadas e transferidas para o próximo estágio. No terceiro estágio as peças são montadas obtendo o produto final. Este problema possui muitas aplicações em indústrias de manufatura e pertence a classe de problemas de otimização combinatória NP-difícil. Para resolução deste problema é realizada uma abordagem mono-objetivo e uma multi-objetivo, onde a primeira visa encontrar uma sequência de *n* tarefas que minimize o atraso total, enquanto a segunda busca encontrar uma sequência de *n* tarefas para minimização simultânea do tempo total de fluxo e do atraso total. Para abordagem mono-objetivo são propostos quatro diferentes algoritmos baseados nas metaheurísticas: o GRASP-RVND, que corresponde a aplicação conjunta das metaheurísticas GRASP (*Greedy Randomized Adaptive Search Procedure*) e RVND (*Random Variable Neighborhood Descent*), o ILS (*Iterated Local Search*), o IG (*Iterated Greedy*) e o ILS-IG (que corresponde a aplicação conjunta das metaheurísticas ILS e IG). Foram realizados experimentos computacionais para comparar a eficiência dos algoritmos e os resultados obtidos são comparados com os resultados de um algoritmo PSA (*Simulated Annealing Híbrido*) da literatura. Nos experimentos foram usadas instâncias de pequeno e grande porte. Os resultados dos algoritmos também foram comparados com os resultados obtidos por um modelo de Programação Linear Inteira (MILP) para instâncias de pequeno porte. Os resultados foram analisados estatisticamente e os testes mostram que os algoritmos propostos foram superiores ao PSA em todas as classes de instâncias. Para a abordagem multiobjetivo são propostos dois algoritmos genéticos híbridos, obtidos a partir da aplicação combinada das metaheurísticas NSGA-II (*Non-dominated Sorting Genetic Algorithm-II*) e IG, sendo que, um deles utiliza uma busca local para melhorar as soluções dominantes. Os algoritmos foram comparados com o algoritmo tradicional NSGA-II. A análise estatística aplicada sobre os resultados obtidos mostra que houve grande melhoria em relação aos resultados gerados pelo NSGA-II.

ABSTRACT

CAMPOS, Saulo Cunha, M.Sc., Universidade Federal de Viçosa, february of the 2015. **Aplicação de metaheurísticas para o problema de programação da produção em ambiente Assembly Flowshop com três estágios e tempos de setup dependentes.** Fugleman teacher: José Elias Claúdio Arroyo.

This paper addresses the Assembly flowshop problem with three stages, where there are *m* parallel machines in the first stage, a transport machine in the second period and an assembly machine in the third stage. In the first stage, different parts of the product are produced independently in parallel machines. In the second stage, the manufactured parts are collected and transferred to the next stage. In the third stage the parts are assembled to give the final product. This problem has many applications in manufacturing industries and belongs to the class of combinatorial optimization problems NP-hard. In order to solvie this problem, mono-objective and multi-objective approach are proposed, where the first aims to find a *n* task sequence that minimizes the total tardiness, while the second attempts to find a sequence for simultaneous minimization of the total flow time and the total tardiness. For the mono-objective approach is proposed four different algorithms are proposed based on metaheuristics: GRASP-RVND, which is the combined application of metaheuristics GRASP (Greedy Randomized Adaptive Search Procedure) and RVND (Random Variable Neighborhood Descent), the ILS (Iterated Local Search), IG (Iterated Greedy) and the ILS-IG (which corresponds to the combined application of the metaheuristics ILS and IG). We performed computational experiments to compare the efficiency of the proposed algorithms, and the obtained results are compared with the results of a PSA (Simulated Annealing Hybrid) literature algorithm. In our experiments we use small and large instances. The results of the algorithms were also compared with the results obtained by an Integer Linear Programming Model (MILP) for small instances. The results were analyzed statistically and the tests show that the proposed algorithms are superior to PSA in all classes of instances. For a multi-objective approach, we propose two hybrid genetic algorithms, obtained from the combined application of the metaheuristics NSGA-II (Non-dominated Sorting Genetic Algorithm-II) and IG. One of these algorithms uses a local search for improve the dominant solutions. The algorithms were compared with the traditional algorithm NSGA-II. The statistical analysis applied to the obtained results shows that there was a great improvement with relation to the results generated by the NSGA-II.

1 INTRODUÇÃO

1.1 Considerações iniciais

Segundo Pinedo (2005) a Programação da Produção é um processo decisório utilizado regularmente por muitas indústrias e empresas de serviço, para alocação de recursos limitados em atividades diversas do processo de negócio. Tal alocação deve ser feita de forma a otimizar os objetivos e metas pretendidos pela empresa. Ainda segundo Pinedo (2005), estes recursos podem variar de acordo com tipo de segmento, como por exemplo, máquinas em uma indústria, pistas de pouso em um aeroporto, programas de computador a serem executados, etc. Da mesma forma os objetivos também podem variar, podendo ser a minimização do tempo total de conclusão das tarefas, diminuição do tempo médio de execução, redução de atrasos, etc.

Este trabalho aborda o problema de programação de tarefas em máquinas (Shen et al., 2006). Este problema, geralmente, consiste em encontrar o melhor sequenciamento de um conjunto de *n* tarefas (ou produtos) que devem ser executadas em um conjunto de *m* máquinas (organizadas em um ambiente de produção) respeitando algumas condições realcionadas a tarefas e máquinas. O melhor sequenciamento de tarefas corresponde a uma solução que minimiza ou maximiza um determinado critério (função objetivo).

Problemas de programação (scheduling) da produção têm sido amplamente estudados na literatura desde os anos 50. Geralmente, estes problemas envolvem: determinar qual tarefa estará associada a qual máquina e determinar uma ordem de processamento das tarefas em cada uma das máquinas, de forma que se otimize algum fator envolvido no sistema produtivo. A programação da produção geralmente visa otimizar um ou vários objetivos simultaneamente, os quais podem estar associados a utilização eficiente dos recursos, como por exemplo, diminuição do tempo total da produção, atendimento às datas de entrega combinadas com os clientes, diminuição dos custos de produção, diminuição de estoques, dentre outros.

A programação da produção é de grande interesse para indústrias de manufatura por ser, fundamentalmente, um processo de otimização, ou seja, é possível aumentar a produtividade ou lucratividade sem necessariamente investir em novos equipamentos e aumento do quadro de funcionários. Em um mercado comercial globalizado e competitivo, a otimização do processo produtivo e a alocação eficiente de recursos pode ser um grande diferencial.

Entre as diversas classes de problemas de programação da produção de manufatura descritos na literatura, existem aquelas que lidam com montagem de produtos (também conhecido como sincronização de produtos). De acordo com Pinedo (2005) esta classe de problema está relacionada a ambientes onde diversas peças (ou matérias-primas) produzidas ou adquiridas devem chegar, aproximadamente, ao mesmo tempo no local de montagem para que o produto final seja montado ou terminado.

Neste tipo de problema, o processo produtivo é dividido em etapas, originalmente, em duas etapas (POTTS ET AL., 1995), onde a primeira etapa corresponde à fabricação (ou aquisição) independente das diversas peças que compõem um produto e, a segunda etapa, corresponde a montagem do produto final. Porém, a quantidade de estágios antes do estágio de montagem pode variar de acordo com o processo do segmento industrial abordado, como por exemplo, no trabalho de Koulamas e Kyparisis (2001) considera-se o problema com três estágios. Para tornar o trabalho mais realístico, esses autores inseriram um estágio intermediário entre o estágio de fabricação e o estágio de montagem. Neste estágio são transportadas as peças produzidas para o setor de montagem. Assim, o processo de produção possui três estágios: fabricação, transporte e montagem.

Outra característica a ser destacada no problema é que, entre os estágios, a transferência de um estágio para o outro obedece a regra de um ambiente *flowshop*, ou seja, um dado estágio só pode iniciar após todos os outros estágios anteriores terem sido executados e concluídos. Por este motivo, estes problemas são conhecidos na literatura como *Assembly Flowshop Scheduling* (*AFS*). Há variações do AFS em relação a quantidade de estágios, sendo dois estágios, denotado por 2sAFS, e com três estágios, denotado por 3sAFS.

Neste trabalho é abordado o problema 3sAFS com tempos de preparação dependentes da sequência de tarefas em cada estágio, ou seja, é considerado que há um determinado tempo para preparar uma máquina para o processamento de uma tarefa *j* após o término de uma tarefa anterior *j-1*. Abordagens com tempos de preparação (ou tempos de *setup*) são comuns na literatura e tornam o problema ainda mais realístico. Na prática, estes tempos correspondem ao tempo necessário para preparar ferramentas, configurar uma máquina, posicionar equipamentos, efetuar limpeza de componentes, dentre outras operações comuns no dia a dia das empresas de manufatura. Neste contexto, existem essencialmente dois tipos de tempo de preparação: os dependentes da sequência das tarefas e os não dependentes (independentes) da sequência das tarefas. No segundo tipo, o tempo de preparação está associado a cada tarefa, já

no primeiro tipo, o tempo de preparação depende também da tarefa processada anteriormente, isto é, o tempo de preparação está associado a cada par de tarefas (j-1, j). O tempo de preparação dependente da sequência pode ser encontrado em operações de pinturas, por exemplo, o esforço para limpar uma máquina que utilizou uma cor escura é maior que a limpeza de uma cor clara. Assim, o tempo de preparação de uma tarefa j pode ser uma unidade de tempo, caso a tarefa j-1 (tarefa antessora) tenha sido pintada com uma cor clara; caso contrário, se uma cor escura foi utilizada, o tempo de processamento será maior que uma unidade de tempo.

Na literaratura há diversas abordagens sobre o problema 2sAFS, como por exemplo, os trabalhos de (AL-ANZI & ALLAHVERDI, 2007), (NAVAEI ET AL., 2013) e (ALLAHVERDI & AYDILEK, 2014), e também a respeito do problema 3sAFS, como os trabalhos de (MABOUDIAN & SHAFAEI, 2009), (HATAMI ET AL., 2010) e (ANDRÉS & HATAMI, 2011).

1.2 O problema e a sua importância

Como já citado anteriormente, o problema aqui abordado é o 3sAFS com tempos de preparação dependentes da sequência. De forma mais detalhada, o ambiente produtivo abordado é composto por um primeiro estágio, composto por m máquinas paralelas e independentes, onde as peças (ou componentes) dos produtos são fabricadas. Após a fabricação, o segundo estágio é iniciado. Ele é formado por uma máquina simples, que coleta as peças do estágio 1 e as transfere para o terceiro estágio (estágio de montagem). O terceiro estágio é também formado por 1 máquina simples que faz a montagem das peças, transformando-as em um produto completo e acabado.

Para realizar uma tarefa j é realizado um conjunto de $m+2$ operação em um ambiente formado por $m+2$ máquinas, sendo que m operações são realizadas de forma independente por uma máquina no primeiro estágio (fabricação das m peças da tarefa). As outras duas operações são de transporte e montagem feitas, respectivamente, nos estágios 2 e 3. A Figura 1 ilustra esse processo.

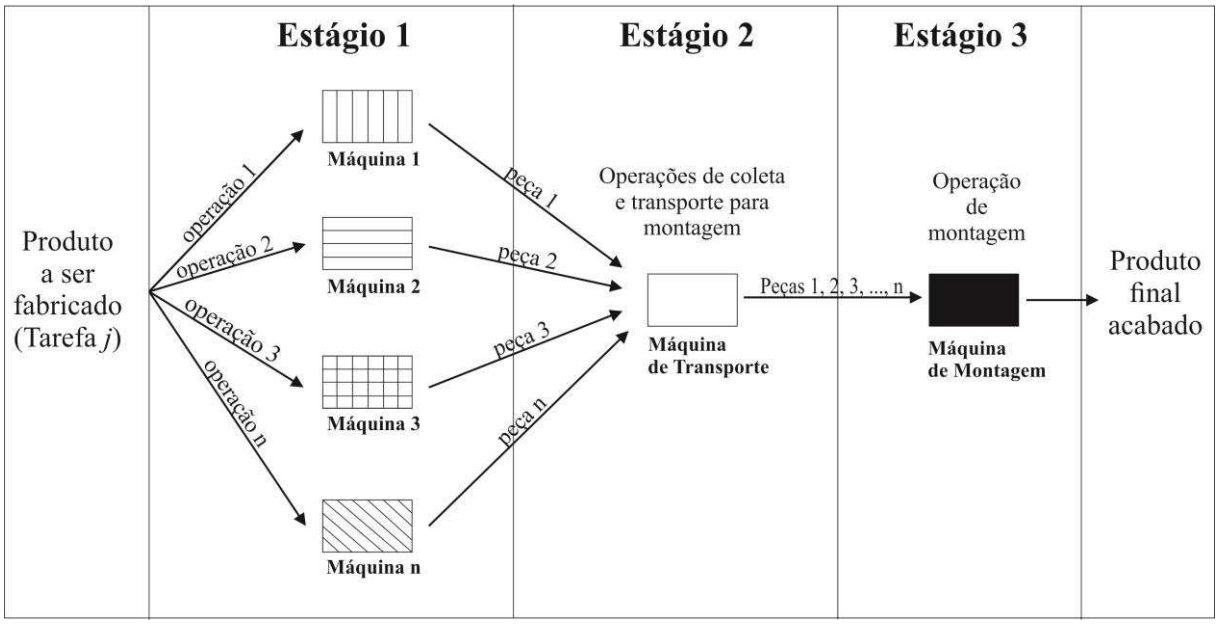

Figura 1: Ambiente de produção 3sAFS com máquinas paralelas no estágio 1 e máquinas simples nos estágios 2 e 3

Koulamas e Kyparisis (2001) afirmam que este tipo de ambiente de produção surgiu em decorrência da pressão do mercado sobre as empresas, para que ofereçam uma variedade maior de produtos e customizações. Diversas indústrias de manufatura possuem problemas semelhantes a estes, como por exemplo, indústrias de placas de circuito (PINEDO, 2005), indústrias de fertilizantes (GERODIMOS ET AL., 2000), indústrias de automóveis (LEE e CHENG, 1993), indústrias de móveis (NAVAEI ET AL., 2013) dentre diversos outros segmentos, também abordados na literatura, como por exemplo, sistema de gestão de cadeia de suprimentos (LIN ET AL., 2007), sistemas de impressão (ZHANG ET AL., 2010) e sistemas de banco de dados distribuídos (ALLAHVERDI & AL-ANZI, 2006).

Na literatura de problemas de programação de tarefas, os objetivos a serem otimizados são vários. Dentre eles, tem-se a minimização do tempo final de produção (*makespan*), minimização do tempo de fluxo da produção, minimização de atrasos e adiantamentos na entrega, minimização do atraso máximo de entrega, entre outros. A otimização desses objetivos leva a redução de custos empresariais e, consequentemente, ao aumento da lucratividade. Em um ambiente comercial globalizado e competitivo como o atual cenário mundial, a otimização de qualquer um destes objetivos pode ser um considerada um fator de grande contribuição para o sucesso de uma empresa.

Este trabalho aborda o problema 3sAFS com tempos de preparação dependentes da sequência em todos os estágios, fato que ainda não foi visto na literatura. Este fator deixa o problema mais realístico, pois a preparação de máquinas é algo comum na manufatura, mas ao

mesmo tempo, deixa o problema ainda mais complexo, aumentando o tamanho do espaço de soluções (número de possibilidades de sequenciamento). Outro importante ponto a ser destacado, é que os algoritmos heurísticos aqui propostos são capazes de resolver problemas com até 200 tarefas, fato que também não foi encontrado na literatura.

Além disso, neste trabalho, são desenvolvidos métodos heurísticos para minimização de um único objetivo (o atraso total) e, também, métodos de otimização multi-objetivo para a minimização de dois objetivos simultaneamente (atraso total e tempo total de fluxo). Os experimentos computacionais realizados mostram que os métodos aqui propostos foram estatisticamente superiores a outros métodos da literatura.

Ademais da importância prática na vida real, outro fator que motiva este trabalho é a investigação sobre a resolução de problemas de natureza combinatória NP-Díficil. Para problemas da classe NP-Difícil é sabído que não se conhecem algoritmos eficientes para sua resolução exata (LEVITIN, 2003). Assim, são necessárias outras técnicas de resolução, como técnicas aproximadas, produzidas através do uso de metaheurísiticas (TALBI, 2009). A construção de algoritmos heurísticos eficientes é algo de grande interesse e constantemente estudado nas ciências computacionais e afins. Deste ponto de vista, este trabalho oferece contribuições importantes, pois apresenta, a modelagem matemática do problema, diversos algoritmos heurísticos, técnicas de busca local e métodos híbridos. Além disso, aborda o uso de experimentos e técnicas estatísticas para a qualificação dos resultados.

1.3 Hipótese

O problema aqui abordado é de otimização combinatória pertencente à classe NP-Difícil. Para resolução dessa classe de problemas, em tempo aceitável, devem ser utilizados métodos aproximados (heurísticas), os quais não oferecem garantias de que a melhor solução sempre será encontrada, mas geram soluções próximas, de boa qualidade.

Dessa forma, este trabalho baseia-se na hipótese de que é possível encontrar bons resultados para o problema 3sAFS utilizando algoritmos heurísticos, baseados em metaheurísticas, em um tempo computacional aceitável.

1.4 Objetivos

O objetivo geral deste trabalho é desenvolver abordagens heurísticas eficientes mono-objetivo (minimização de um único objetivo) e multi-objetivo (minimização de mais de um

objetivo) para o problema 3sAFS, com tempos de preparação dependentes da sequência, baseado em metaheurísticas.

Os objetivos específicos deste trabalho são:

a) Apresentar um levantamento bibliográfico sobre o problema abordado e os métodos utilizados para resolvê-los;
b) Definir a fórmula de cálculo dos diversos objetivos aqui abordados;
c) Obter ou implementar algoritmos da literatura para comparação com as propostas aqui realizadas;
d) Propor algoritmos heurísticos mono-objetivo para resolução do problema;
e) Propor algoritmos heurísticos multi-objetivos para resolução do problema;
f) Realizar testes e experimentos de calibração, a fim de encontrar ajustes finos para os algoritmos propostos;
g) Realizar experimentos computacionais de comparação entre as propostas deste trabalho e as já existentes na literatura;
h) Analisar estatisticamente os resultados obtidos.

1.5 Organização da dissertação

Esta dissertação foi elaborada de acordo com um dos formatos recomendados pela Comissão do Programa de Pós-Graduação em Ciência da Computação da UFV: coletânea de artigos. Neste formato, a pesquisa é apresentada através de um conjunto de artigos.

Até o momento esta pesquisa gerou três publicações de artigos:

- **Campos *et al.* (2013)**: apresenta a proposta de uma heurística híbrida, denominada GRASP-RVND, que utiliza a metaheurística GRASP para construir novas soluções iterativamente e uma variação da metaheurística VND, como método de busca local, para minimizar a função linear dada pela soma ponderada do atraso máximo e do fluxo médio das tarefas. Além disso, é feita uma comparação com Hatami et al (2010) e as estatísticas sugerem que GRASP-RVND obtém melhores resultados para o problema, em um mesmo tempo de execução.
- **Campos e Arroyo (2014a)**: apresenta uma abordagem multi-objetivo para minimização simultânea do tempo total de fluxo e atraso máximo. Neste caso,

compara-se três algoritmos heurísticos: o NSGA-II, a heurística mono-objetivo GRASP-RVND (citada anteriormente) e uma versão do NSGA-II, que intesifica a exploração das soluções não dominadas através da metaheurística *Iterated Greedy*, denominada NSGA2_IG.

- **Campos e Arroyo (2014b)**: É realizada uma abordagem mono-objetivo para minimização do atraso total, utilizando as metaheurísticas ILS e *Iterated Greedy* combinadas (ILS-IG). Três algoritmos são avaliados e comparados com o GRASP-RVND citado anteriormente, além de serem apresentados diversos experimentos computacionais para configuração dos algoritmos.

Apesar das publicações, optamos por escrever dois novos artigos, sintetizando todas as abordagens efetuadas ao longo do desenvolvimento da pesquisa, oferecendo informações mais detalhadas aos leitores. Entendemos que, assim, podemos proporcionar um maior entendimento dos assuntos aqui tratados. Os artigos aqui apresentados deverão ser submetidos à publicações em revistas ou congressos especializados. Sendo assim, organiza-se esta dissertação em dois artigos, correspondentes aos capítulos 2 e 3 (abordagens mono-objetivo e multi-objetivo, respectivamente).

No primeiro artigo, sintetizamos toda a abordagem mono-objetivo realizada para minimização do atraso total. São apresentados quatro diferentes algoritmos heurísticos, experimentos computacionais para configuração e comparação com um algoritmo da literatura. Já no segundo artigo, é descrita a abordagem multi-objetivo para minimização simultânea do atraso total e do tempo total de fluxo. Neste artigo, utiliza-se um conhecido algoritmo da literatura multi-objetivo, o NSGA-II, combinado com a metaheurística *Iterated Greedy*, em duas diferentes versões. A abordagem ainda contempla experimentos computacionais para calibração, comparação e análises estatísticas dos resultados.

No capítulo 4, são apresentados as conclusões gerais do trabalho. As referências bibliográficas da parte introdutória da dissertação são listadas a seguir.

1.6 Referências bibliográficas do capítulo 1

Al-Anzi, F. S., & Allahverdi, A. (2006). A hybrid tabu search heuristic for the two-stage assembly scheduling problem. International Journal of Operations Research, 3(2), 109-119.

Al-Anzi, F. S., & Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. European Journal of Operational Research, 182(1), 80-94.

Andrés, C., & Hatami, S. (2011, September). The three stage assembly permutation flowshop scheduling problem. In V international conference on industrial engineering and industrial management (pp. 867-875).

Allahverdi, A., & Al-Anzi, F. S. (2006). A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. Computers & Operations Research, 33(4), 1056-1080.

Allahverdi, A., & Aydilek, H. (2014). The two stage assembly flowshop scheduling problem to minimize total tardiness. Journal of Intelligent Manufacturing, 1-13.

Campos, S. C., Arroyo, J. E. C., & Gonçalves, L. B. (2013). Uma heuristica GRASP-VND para o problema de sequenciamento de tarefas num ambiente assembly flowshop com três estágios e tempos de setup dependentes da sequência. In Proceedings of the XLV Brazilian Symposium of Operational Research.(Natal-RN, Brazil, Setember 16-19, 2013).

Campos, S. C., & Arroyo, J. E. C. (2014, July). NSGA-II with iterated greedy for a bi-objective three-stage assembly flowshop scheduling problem. In Proceedings of the 2014 conference on Genetic and evolutionary computation (pp. 429-436). ACM.

Campos, S. C., & Arroyo, J. E. C. (2014) Uma heuristica ILS-IG para o problema de sequenciamento de tarefas num ambiente assembly flowshop com três estágios e tempos de setup dependentes da sequência. In Proceedings of the XLVI Brazilian Symposium of Operational Research.(Salvador-BA, Brazil, Setember 16-19, 2014).

Gerodimos, A. E., Glass, C. A., & Potts, C. N. (2000). Scheduling the production of two-component jobs on a single machine. European Journal of Operational Research, 120(2), 250-259.

Hatami, S., Ebrahimnejad, S., Tavakkoli-Moghaddam, R., & Maboudian, Y. (2010). Two meta-heuristics for three-stage assembly flowshop scheduling with sequence-dependent setup

times. The International Journal of Advanced Manufacturing Technology, 50(9-12), 1153-1164.

Koulamas, C., & J Kyparisis, G. (2001). The three-stage assembly flowshop scheduling problem. Computers & Operations Research, 28(7), 689-704.

Lee, C. Y., Cheng, T. C. E., & Lin, B. M. T. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. Management Science, 39(5), 616-625.

Levitin, A. (2003) Introduction to the design & analysis of algorithms. Addison-Wesley Reading, MA.

Lin, B. M. T., Cheng, T. C. E., & Chou, A. S. C. (2007). Scheduling in an assembly-type production chain with batch transfer. Omega, 35(2), 143-151.

Maboudian, Y. & Shafaei, R. (2009). Modeling a bi-criteria two stage assembly flow shop scheduling problem with sequence dependent setup times. In Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management (Hong Kong, China, December 8-11, 2009), 1748-1752.

Navaei, J., Ghomi, S. F., Jolai, F., Shiraqai, M. E., & Hidaji, H. (2013). Two-stage flow-shop scheduling problem with non-identical second stage assembly machines. The International Journal of Advanced Manufacturing Technology, 69(9-12), 2215-2226.

Pinedo, M. (2005). Planning and scheduling in manufacturing and services(Vol. 24). New York: Springer.

Potts, C. N., Sevast'Janov, S. V., Strusevich, V. A., Van Wassenhove, L. N., & Zwaneveld, C. M. (1995). The two-stage assembly scheduling problem: complexity and approximation. Operations Research, 43(2), 346-355.

Shen, W., Wang, L., e Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. Systems, Man, and Cybernetics, Part C: Applications and Reviews. IEEE Transactions on, vol. 36 no.4, 563–577.

Talbi, E. G. (2009). Metaheuristics: from design to implementation (Vol. 74). John Wiley & Sons.

Zhang, Y., Zhou, Z., & Liu, J. (2010). The production scheduling problem in a multi-page invoice printing system. Computers & Operations Research, 37(10), 1814-1821.

2 ALGORITMOS HEURÍSTICOS PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM UM AMBIENTE ASSEMBLY FLOWSHOP COM TRÊS ESTÁGIOS E TEMPOS DE SETUP DEPENDENTES DA SEQUÊNCIA.

2.1 Introdução

Os problemas de programação (*scheduling*) são problemas de tomada de decisão comumente encontrados em sistemas de manufatura, o que mostra sua importância prática e teória (ALLAVERDI ET AL., 2008). Neste tipo de problema, pretende-se encontrar a melhor sequência de operações de manufatura para minimização (em alguns casos, maximização) de um ou mais critérios (objetivos) (SHEN *ET AL.*, 2006).

Estes problemas são amplamente estudados na literatura, sendo que os primeiros trabalhos são remotos da década de 50 (ALLAHVERDI *ET AL.*, 2008). A motivação para investigação deste tipo de problema vem de dois aspectos. O primeiro está relacionado com a importância tática para os negócios: segundo Pinedo (2005) e Lustosa et al. (2008), a programação da produção evolui devido a alta concorrência de mercado e a globalização que pressionam as empresas a reduzirem seus custos. O segundo aspecto está relacionado com o fato da maioria destes problemas pertencerem à classe de problemas NP-Difícil, fazendo com que diversos autores direcionem suas pesquisas para a produção de algoritmos de alto desempenho computacional para resolvê-los. Na literatura existem diversos trabalhos de revisões sobre este assunto, como exemplo: Ruiz e Vázquez-Rodríguez (2010) e Ribas et al. (2010).

Neste trabalho é abordado um subproblema da programação da produção conhecido como "*Assembly Flowshop Scheduling* – AFS", que é um ambiente de produção onde diversas partes do produto (peças e componentes), são fabricadas de forma independente em diferentes linhas de produção e, em seguida, são transferidas para um local onde é realizada a montagem do produto que, geralmente, é feita em uma única máquina. Este ambiente é encontrado, por exemplo, em indústrias que fabricam placas de circuitos (PINEDO, 2005). Nestas indústrias, diversos componentes são fabricados de forma independente por máquinas paralelas em um primeiro estágio (às vezes, até mesmo em fábricas diferentes) e, depois, em um segundo estágio, são agrupados e soldados às placas, formando o produto final. Koulamas e Kyparisis (2001)

afirmam que este tipo de ambiente de produção cresce em resposta à pressão do mercado sobre as empresas, para que ofereçam uma variedade maior de produtos e customizações.

Na literatura, o problema AFS é abordado considerando dois (2sAFS) ou três (3sAFS) estágios. Ambos os casos pertecem à classe de problemas NP-Difícil, sendo o último abordado pelo presente trabalho. O 3sAFS consiste em executar (ou produzir) *n* tarefas (ou produtos) em três etapas. Na primeira etapa (estágio 1), diferentes partes de uma tarefa (produto) são processadas em *m* máquinas paralelas independentes. Já na segunda etapa (estágio 2), conhecida como etapa de transporte, as partes produzidas são coletadas e transferidas para uma linha de montagem, que corresponde à terceira etapa (estágio 3), onde o produto final é montado. Tanto na segunda quanto na terceira etapa (estágios 2 e 3), as tarefas são processadas por uma máquina simples.

Já o 2sAFS não possui a etapa de transporte. Dessa forma, imediatamente após a etapa de produção no estágio 1, o produto final é montado em um segundo estágio. Esta é a única diferença entre eles. Porém Koulamas e Kyparisis (2001) afirmam que o modelo com três estágios é mais realista e o consideram como uma evolução do modelo com dois estágios.

Potts et al. (1995) foram uns dos primeiros a abordar o problema AFS. Eles provaram que o problema é NP-Difícil e utilizaram métodos heurísticos baseados em emuneração implícita para minimização do *makespan*, que consiste em minimizar o tempo total da produção. Mais a frente, Hariri e Potts (1997) abordaram o mesmo problema. Eles propuseram um limite inferior e diversas relações de dominância, além de apresentar um algoritmo *branch-and-bound* incorporando as relações limite inferior e de dominância. Tozkapan *et al.* (2003) também propuseram um *branch-and-bound* e regras de dominância; porém, consideraram como função objetivo a média ponderada do tempo total de fluxo. Allahverdi e Al-Anzi (2006) também abordaram o problema para minimização do *makespan* através de duas heurísticas evolucionárias e incorporaram ao problema tempos de preparação não dependentes da sequência. Já Al-Anzi e Allahverdi (2007) propuseram a minimização do atraso máximo das tarefas, também considerando tempos de *setup* não dependentes da sequência, através da aplicação de diversas metaheurísticas evolucionárias. Allahverdi e Al-Anzi (2008) propuseram várias heurísticas para resolução do problema com minimização da função linear ponderada do *makespan* e tempo médio de fluxo. Em 2009, Allahverdi e Al-Anzi voltaram a abordar o problema para minimização do tempo total de fluxo através de heurísticas. Mais tarde, Torabzadeh e Zandieh (2010) e Shokrollahpour et al. (2011) desenvolveram novos algoritmos

para o mesmo problema e função objetivo de Allahverdi e Al-Anzi (2008) e, após comparações, os resultados apresentados foram melhores. Seidgar *et al.* (2013) consideraram um algoritmo competitivo imperialista (ICA) para minimizar a função linear ponderada para os seguintes objetivos: *makespan* e tempo médio de fluxo. Yan *et al.* (2014) também abordaram o problema utilizando função linear ponderada, porém, com a proposta de minimizar o *makespan* e os custos de atrasos e adiantamentos. Além disso, eles utilizaram a metaheurística VNS combinada com algoritmos *electromagnetism-like*. Já Allahverdi e Aydilek (2014) propuseram diversos algoritmos baseados nas metaheurísticas *Simulated Annealing* e Algoritmo Genético, combinados com um algoritmo de inserção (PIA), com intuito de minimizar o atraso máximo.

Nós últimos anos, abordagens sobre variações do 2sAFS têm sido apresentadas. Fattahi *et al.* (2013) abordaram uma variação do 2sAFS, onde o ambiente de produção no primeiro estágio é um *flowshop* híbrido. Eles propuseram um modelo matemático e várias heurísticas baseadas no algoritmo de Johnson para minimização do *makespan*. Já Mozdgir *et al.* (2012) abordaram uma variação do problema que considera múltiplas máquinas não idênticas no estágio de montagem. Eles fazem a minimização da soma ponderada de *makespan* e tempo médio de conclusão, com tempos de preparação dependentes da sequência no primeiro estágio. Além da resolução através da programação linear mista, eles propuseram uma heurística VNS híbrida para resolução de instâncias maiores. Navaei (2013) também abordou o problema com várias máquinas de montagem não idênticas; porém, buscou a minimização do *makespan*, através da aplicação da metaheurística *Simulated Annealing*. Além disso, foi apresentado um caso de estudo real da aplicação do método em uma indústria de manufatura de móveis.

O 3sAFS foi abordado pela primeira vez por Koulamas e Kyparisis (2001). Os autores analisaram diversas heurísticas construtivas e propuseram a minimização do *makespan*. Hatami et al (2010) foram os primeiros a tratar o 3sAFS com tempos de *setup* dependentes da sequência das tarefas, porém, aplicado somente às máquinas do primeiro estágio. Eles usaram as metaheurísticas *Simulated Annealing* e Busca Tabu para minimizar a função linear dada pela soma ponderada dos objetivos: atraso máximo e fluxo médio das tarefas. Andrés e Hatami (2011) propuseram um modelo matemático MIP para minimização do *makespan*, considerando tempos de *setup* no primeiro e terceiro estágios. Maleki *et al.* (2012) consideraram o 3sAFS com bloqueio e tempos de setup dependentes da sequência das tarefas, mas com o objetivo de minimizar a função linear ponderada com a combinação dos objetivos, tempo médio de conclusão e *makespan*, utilizando um algoritmo baseado na metaheurística *Simulated Annealing*. Já em 2013, Campos *et al.* (2013) abordaram o mesmo problema considerado por

Hatami et al (2010) e propuseram uma abordagem baseada na combinação das metaheurísticas GRASP e RVND, obtendo melhoria nos resultados finais.

Recentemente têm surgido abordagens multiobjetivos para 3sAFS. Nestas abordagens, mais de um objetivo são minimizados simultaneamente, como, por exemplo, os trabalhos de Tajbakhsh *et al.* (2013) e Campos e Arroyo (2014). O primeiro abordou o problema com a utilização de algoritmo genético e otimização por enxame de partículas (PSO) para minimização do *makespan* e a soma de custos de adiantamentos e atrasos, enquanto o segundo propõe a utilização do tradicional algoritmo para resolução de problemas multiobjectivo NSGA2 (*non dominated sorting genetic algorithm* II) combinado com o algoritmo IG (*iterated greedy*), para minimização do tempo total de conclusão e atraso total.

Diferentemente da maioria dos trabalhos da literatura e com o intuito de deixar o problema ainda mais realístico, consideramos tempos de *setup* dependentes da sequência em todos os estágios. A proposta aqui é resolver o problema pela aplicação combinada dos algoritmos ILS – *Iterated Local Search* – (LOURENÇO *ET AL.*, 2002) e IG – *Iterated Greedy* (RUIZ e STÜTZLE, 2007) – com o intuito de minimizar o atraso total das tarefas.

O ILS é uma metaheurística simples, que a partir de uma solução inicial, aplica buscas locais de forma iterativa, e, para impedir que o algoritmo fique estagnado em ótimos locais, utiliza a estratégia de perturbar a solução corrente (HOOS e STÜTZLE, 2004). Naderi et al. (2010) afirma que o ILS tem sido amplamente utilizado para construir soluções do estado-da-arte para problemas de sequenciamento de tarefas em ambientes *flowshop*. O IG também aplica buscas locais iterativamente; porém, para que não fique preso em ótimos locais, usa uma estratégia de destruição-construção da solução. Ruiz e Stützle (2008) aplicaram o IG com grande sucesso para problemas em ambiente *flowshop* com tempos de *setup* dependentes da sequência e demostraram que tal algoritmo, apesar de sua simplicidade, produz resultados de estado-da-arte.

Os detalhes destes algoritmos são vistos nas próximas seções do artigo, que está estruturado na seguinte forma: a seção 2.2 apresenta a definição formal do problema abordado, a seção 2.3 apresenta as soluções heurísticas supracitadas, a seção 2.4 descreve os experimentos computacionais realizados, e, por fim, na seção 2.5, encontram-se as conclusões.

2.2 Definição formal do problema

O problema investigado neste trabalho é o sequenciamento de tarefas em um ambiente *Assembly Flowshop* com três estágios. Este problema consiste em processar (ou fabricar) um conjunto de *n* tarefas (produtos), sendo que cada tarefa possui *m* componentes (ou peças) que são fabricados de forma independente. No primeiro estágio, os componentes das tarefas são fabricados em um ambiente de *m* máquinas paralelas. No segundo estágio, as peças de uma tarefa são transportadas para uma máquina onde é feito a montagem do produto, constituindo o terceiro estágio do sistema produtivo. Os estágios 2 e 3 são constituídos por máquinas simples diferentes. Todas as máquinas processam apenas uma tarefa por vez e não podem ser interrompidas durante seu processamento. Para cada tarefa existem *m+2* operações distintas, sendo que *m* operações independentes são feitas no primeiro estágio e as outras duas operações são realizadas nos dois últimos estágios, respectivamente.

Os três estágios caracterizam o problema como um ambiente de produção *flowshop*. Ou seja, uma tarefa *j* só pode iniciar seu processamento no estágio 2 (transporte) quando todas suas peças são finalizadas no estágio 1, e o estágio 3(montagem) só poderá ser iniciado após a tarefa *j* ser totalmente finalizada no estágio 2.

O sequenciamento das *n* tarefas nas máquinas é representado por uma permutação simples (sequência), que denota a ordem de execução das tarefas nas máquinas e nos estágios. Todas as tarefas estão disponíveis para serem processadas no tempo zero e nenhuma máquina pode processar mais de uma tarefa por vez.

As notações dos dados de entrada do problema são mostradas a seguir:

- n – denota o número de tarefas;
- m – denota o número de máquinas no primeiro estágio;
- i, j – denotam as tarefas ($i, j = \{0, 1, 2, \ldots, n\}$, sendo que 0 representa uma tarefa fictícia);
- h – corresponde a uma posição na sequência;
- k – denota uma máquina no primeiro estágio ($k = \{1, 2, \ldots, m\}$);
- $t_{k,j}$ – denota o tempo de processamento da tarefa *j* na máquina *k*, no estágio 1;
- tt_j – denota o tempo de processamento da tarefa *j* na máquina de transporte do estágio 2;

- ta_j – denota o tempo de processamento da tarefa *j* na máquina de montagem do estágio 3;
- $s1_{k,j-1,j}$ – denota o tempo de preparação (*setup*) da máquina *k*, no estágio 1, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1, ..., *n*);
- $s2_{j-1,j}$ – denota o tempo de preparação (*setup*) da máquina de transporte, no estágio 2, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1, ..., *n*);
- $s3_{j-1,j}$ – denota o tempo de preparação (*setup*) da máquina de montagem, no estágio 3, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1,..., *n*);
- d_j – denota a data de entrega da tarefa *j*.

O objetivo do problema é determinar o sequenciamento das tarefas que minimize o atraso total das tarefas (*T*), calculado pela seguinte equação:

$$T = \sum_{j=1}^{n} A_j$$

Onde A_j corresponde ao atraso da tarefa *j* e é calculado da seguinte maneira:

$$A_j = max\{0, C_{3_j} - d_j\}$$

C_{3_j}, C_{2_j} e C_{1_j} representam os tempos de conclusão da tarefa *j*, respectivamente, nos estágios 3, 2 e 1. Eles são calculados pelas seguintes equações:

$$C_{3_j} = \max\left\{C_{2_j}, \quad C_{3_{j-1}}\right\} + s3_{j-1,j} + ta_j$$

$$C_{2_j} = max\left\{C_{1_j}, \quad C_{2_{j-1}}\right\} + s2_{j-1,j} + tt_j$$

$$C_{1_j} = \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} s1_{k,i-1,i} + t_{j,k}\right\} + C_{1_{j-1}}$$

A Figura 2 mostra um exemplo do problema para *n*=5 tarefas e *m*=2 (M1 e M2) máquinas paralelas no estágio 1. As máquinas de transporte e montagem são, respectivamente, *Mt* e *Ma*. Os blocos pretos denotam os tempos de *setup*. As datas de entrega das tarefas são *d*1 = 24, *d*2 = 15, *d*3 = 35, *d*4 = 35 e *d*5 = 24. Os tempos de processamento e os tempos de *setup* podem ser vistos na Figura 2. A solução em questão é dada pela sequência S = {*j2*, *j1*, *j4*, *j3*,

j5}. A Figura 2 também apresenta os tempos de conclusão de cada tarefa e seus atrasos. O valor da função objetivo é *T* = (0+5+0+5+15) = 25.

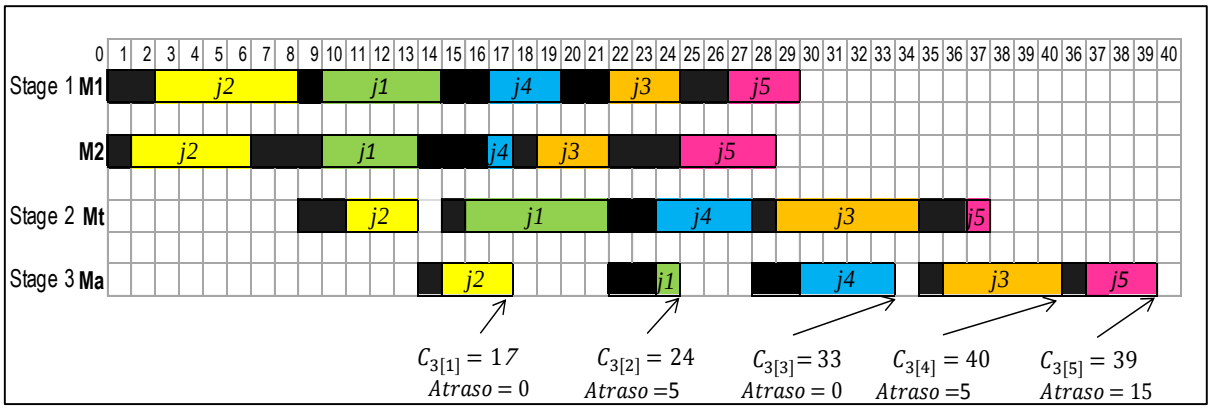

Figura 2: Gráfico de Gantt para o sequenciamento da tarefas {j2, j1, j4, j3, j5}

2.2.1 Modelo matemático

Um modelo matemático de programação inteira mista (MILP) para o problema 3sAFS é apresentado. O modelo proposto é baseado no trabalho de Andrés e Hatami (2011), que propôs um modelo matemático não linear (MIP), para minimização do *makespan* e tempos de preparação no primeiro e no terceiro estágio. Aqui foram feitas adaptações para minimização do atraso total, para considerar tempos de preparação em todos os estágios e também para linearização do modelo.

Os dados de entrada já foram apresentados anteriormente na seção 2.2. Já as variáveis de decisão são: *T* (atraso total), C_{3_j} (tempo de conclusão no estágio 3), C_{2_j} (tempo de conclusão no estágio 2) e C_{1_j} (tempo de conclusão no estágio 1). Além delas, também são necessárias ao modelo uma variável (de decisão) binária $X_{j,h}$, igual a 1 se a tarefa *j* ocupar a posição *h* na seqüência, caso contrário, *x* é igual zero.

A seguir é apresentado o modelo não linear para o problema.

Função objetivo: $\min T = \sum_{j=1}^{n} A_j$ (1)

sujeito a:

$$A_h \geq C_{3_h} - \sum_{j=1}^{n} (d_j X_{j,h}) ; \qquad \forall h = 1, \dots, n \quad (2)$$

$$\sum_{h=1}^{n} x_{j,h} = 1; \qquad \forall j = 1, \dots, n \quad (3)$$

$$\sum_{j=1}^{n} x_{j,h} = 1; \qquad \forall h = 1, \dots, n \quad (4)$$

$$C_{1\,1,k} \geq \sum_{j=1}^{n} \left[\left(s1_{k,0,j} + t_{k,j} \right) \times X_{j,1} \right]; \qquad \forall k = 1, \dots, m \quad (5)$$

$$C_{1_{h,k}} \geq C_{1_{h-1,k}} + \sum_{j=1}^{n} \left(t_{k,j} \times X_{j,h} \right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(s1_{k,i,j} \times X_{i,h-1} X_{j,h} \right); \qquad \begin{matrix} \forall h = 2, \dots, n \\ \forall k = 1, \dots, m \end{matrix} \quad (6)$$

$$C_{2_h} \geq C_{1_{h,k}} + \sum_{j=1}^{n} \left(tt_j \times X_{j,h} \right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(s2_{i,j} \times X_{i,h-1} X_{j,h} \right); \qquad \begin{matrix} \forall h = 2, \dots, n \\ \forall k = 1, \dots, m \end{matrix} \quad (7)$$

$$C_{2_1} \geq C_{1_{1,k}} + \sum_{j=1}^{n} \left[\left(s2_{0,j} + tt_j \right) \times X_{j,1} \right]; \qquad \forall k = 1, \dots, m \quad (8)$$

$$C_{2_h} \geq C_{2_{h-1}} + \sum_{j=1}^{n} \left(tt_j \times X_{j,h} \right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(s2_{i,j} \times X_{i,h-1} X_{j,h} \right); \qquad \forall h = 2, \dots, n \quad (9)$$

$$C_{3_h} \geq C_{2_h} + \sum_{j=1}^{n} \left(ta_j \times x_{j,h} \right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(s3_{i,j} \times X_{i,h-1} X_{j,h} \right); \qquad \forall h = 2, \dots, n \quad (10)$$

$$C_{3_1} \geq C_{2_1} + \sum_{j=1}^{n} \left[\left(s3_{0,j} + ta_j \right) \times X_{j,1} \right] \quad (11)$$

$$C_{3_h} \geq C_{3_{h-1}} + \sum_{j=1}^{n} \left(ta_j \times X_{j,h} \right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(s3_{i,j} \times X_{i,h-1} X_{j,h} \right); \qquad \forall h = 2, \dots, n \quad (12)$$

$$C_{1_h} \geq 0; C_{2_h} \geq 0;\ C_{3_h} \geq 0;\ A_h \geq 0; \qquad \begin{matrix} \forall h = 1, \dots, n \\ \forall k = 1, \dots, m \\ \forall j = 1, \dots, n \end{matrix} \quad (13)$$

$$X_{j,h} \in \{0, 1\}; \qquad \begin{matrix} \forall j = 1, \dots, n \\ \forall h = 1, \dots, n \end{matrix} \quad (14)$$

A equação 1 é a função objetivo do modelo que visa a minimização do atraso total *T*. A equação 2 computa o atraso de cada tarefa. As equações 3 e 4 garatem, respectivamente, que

uma posição será alocada somente para uma tarefa e que toda tarefa será alocada a uma posição. As equações 5 e 6 computam os tempos de conclusão para cada uma das máquinas do estágio 1, considerando os tempos de preparação dependentes da sequência. A restrição 7 impõe a relação entre os estágios 1 e 2, de forma que o tempo de conclusão de uma tarefa *j* no estágio 1 seja acrescido ao tempo de conclusão desta tarefa do estágio 2. As restrições 8 e 9 computam os tempos de conclusão do estágio 2, considerando os tempos de preparação dependentes da sequência. De maneira similar à restrição 7, a restrição 10 impõe o relacionamento entre os estágios 2 e 3, e, as equações 11 e 12 calculam o tempo de conclusão do estágio 3, considerando os tempos de preparação dependentes da sequência. Por fim, a restrição 13 representa as condições de não negatividade e a restrição 14 define as variáveis binárias.

É perceptível que, o modelo apresentado é não linear, por causa do termo $X_{i,h-1}X_{j,h}$ nas restrições 6, 7, 9, 10 e 12. Para obter um modelo linear, utilizou-se uma variável auxiliar binária $\Delta_{i,j,h}$ (sendo, $\Delta_{i,j,h} = X_{i,h-1}X_{j,h}$) que substitui o termo não linear citado. Para isso, é necessário adicionar ao modelo as seguintes equações abaixo.

$$\Delta_{i,j,h} - X_{i,h-1} \leq 0 \qquad \begin{matrix} \forall\, h = 2,\ldots,n \\ \forall\, i,j = 1,\ldots,n \end{matrix} \qquad (15)$$

$$\Delta_{i,j,h} - X_{i,h} \leq 0 \qquad \forall\, i,j,h = 1,\ldots,n \qquad (16)$$

$$X_{i,h-1} + X_{j,k} - \Delta_{i,j,h} \leq 1 \qquad \begin{matrix} \forall\, h = 2,\ldots,n \\ \forall\, i,j = 1,\ldots,n \end{matrix} \qquad (17)$$

2.3 Soluções heurísticas propostas

Para problemas da classe NP-Difícil não se conhecem algoritmos de tempo polinomial para a obtenção de soluções ótimas. Os melhores algoritmos, baseados em métodos exatos, para esses problemas, possuem tempo exponencial em função do tamanho da entrada, o que torna inviável a utilização deles (LEVITIN, 2003). Como alternativas de solução, neste trabalho são utilizadas as metaheurísticas, que são métodos aproximados genéricos para geração de soluções de alta qualidade (próxima da ótima) em tempo computacional polinomial (TALBI, 2009).

Neste trabalho avaliamos o desempenho de quatro algoritmos baseados em metaheurísticas, denominados: GRASP-RVND, ILS, IG e ILS-IG. O primeiro (GRASP-RVND) é a aplicação combinada das metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*), proposta por Feo e Resende (1995), juntamente com uma variação da metaheurística VND (*Variable Neighborhood Descent*), proposta por Hansen e Mladenović (2003), que seleciona a estrutura de vizinhança aleatoriamente. Já o ILS corresponde a

aplicação da metaheurística ILS (*Iterated Local Search*) proposta por Lourenço et al. (2002). O IG refere-se ao algoritmo *Iterated Greedy* proposto por Ruiz e Stützle (2007). Por fim, o ILS-IG refere-se a aplicação combinada do ILS e do IG.

2.3.1 GRASP-RVND

O algoritmo GRASP-RVND, proposto nesta seção, é uma heurística que combina as metaheurísticas GRASP e VND. O GRASP é uma metaheurística que iterativamente executa dois métodos: um método de construção (de uma solução) e um método de melhoria (busca local). Já o VND é uma metaheurística baseada na execução sucessiva de buscas locais em diferentes estruturas de vizinhanças. Sendo assim, na heurística proposta, utiliza-se o VND, em sua versão randômica, como método de melhoria do GRASP. Foram executados testes para avaliar o uso do VND tradicional e do RVND (*Random Variable Neighborhood Descent*), sendo que este último é o que gera melhor resultado na resolução do problema, conforme mostrado na seção de experimentos computacionais.

O pseudocódigo da heurística GRASP_RVND é apresentado no Algoritmo 1. O algoritmo é simples e recebe como entrada dois parâmetros: o parâmetro de aleatoriedade α, utilizado no método de construção do GRASP e o parâmetro *Critério_Parada* que define o tempo máximo de execução do algoritmo (critério de parada). Observa-se que o Algoritmo GRASP_RVND, executa iterativamente os métodos *Construção* e *RVND* (linhas 3 e 4). Além disso, a solução *S"* retornada pelo *RVND* é utilizada para atualizar a melhor solução, armazenada em *S* (linhas 5 e 6). O algoritmo segue executando estes passos até que o critério de parada seja satisfeito e a solução *S* seja retornada.

```
Algoritmo 1: GRASP_RVND(α, Critério_Parada)
  Entradas:
    Critério_Parada: Critério de parada do algoritmo;
    α: Fator de aleatoriedade do método de construção;
  Inicio
1   S ← ∅;
2   Enquanto ( Critério_Parada não satisfeito ) faça
3     S' ← Construção( α ); //Construção Gulosa Aleatória
4     S'' ← RVND( S'); //Busca Local
5     Se ( f(S'') < f(S) ) Então
6       S ← S'';
7   Fim_Enquanto
8   Retorna S;
  Fim
```

2.3.1.1 Método de Construção

A fase de construção de uma solução é realizada através da heurística NEH (NAWAZ *ET AL.*, 1983). Nesta heurística, primeiramente, as tarefas são arranjadas de acordo a uma regra de prioridade (ou função gulosa) formando uma lista de tarefas candidatas $J = \{j_1, j_2, ..., j_n\}$. A primeira tarefa j_1 é usada para formar uma sequencia parcial S ($S = \{j_1\}$). Em seguida, escolhe-se a próxima tarefa e é feita a inserção em cada uma das posições de S, obtendo duas sequências parciais $\{j_2, j_1\}$ e $\{j_1, j_2\}$. Estas sequências são avaliadas (calcula-se o valor da função objetivo) e determina-se a melhor. De forma sucessiva, a próxima tarefa j_i da lista é inserida em cada uma das posições da melhor sequência encontrada anteriormente, obtendo-se então, i sequências parciais, onde a melhor é selecionada para a próxima iteração. Quando $i=n$, o método é finalizado e uma sequência (solução) completa (com n tarefas) é obtida e retornada. Ressalta-se que, cada vez que uma tarefa j_i é escolhida de J, ela é removida da lista candidata, ou seja, $J = J - \{j_i\}$, então quando o método é finalizado a lista candidata J é vazia ($J = \varnothing$).

A heurística GRASP tem por característica gerar uma solução diferente a cada iteração. Para que isto ocorra, a heurística NEH é adaptada para, ao invés de selecionar a primeira tarefa da lista J, a tarefa seja selecionada aleatoriamente dentre as t primeiras tarefas. O valor de t é definido de acordo com o tamanho da lista de candidatos e é calculado da seguinte maneira: $t = max\{1, \alpha \times | J |\}$. Note que se $\alpha = 0$, sempre seleciona a primeira tarefa da lista, fazendo com que a escolha seja totalmente gulosa, já se $\alpha = 1$, a escolha será totalmente aleatória, possibilitando que qualquer tarefa seja escolhida. Foram realizados experimentos computacionais (mostrados nos próximos capítulos) para definir a melhor regra de ordenação da lista candidata.

2.3.1.2 Busca Local com RVND

RVND é uma versão adaptada do tradicional VND. Para compreendê-lo é necessário conhecer o VND tradicional. Por isso, nos próximos parágrafos, o funcionamento deste é abordado.

O pseudocódigo do VND tradicional é mostrado no Algoritmo 2.

```
Algoritmo 2: VND(S, nv)
  Início
1   V ←{V1,V2,...,Vnv}; //Ordenar as estruturas de vizinhanças
2   k ← 1;
3   Enquanto (k ≤ nv) faça
4     //Retorna a melhor solução da vizinhança Vk
5     S' ← Melhor_Vizinho(Vk, S);
6     Se ( f(S') < f(S) ) Então
7       S ← S';
8     k ← 1; //reinício
9   Senão
10    k ← k + 1;
11  Fim_Enquanto
12  Retorna S;
  Fim
```

O algoritmo VND possui dois parâmetros de entrada: a solução *S* a ser melhorada e a quantidade *nv* de estruturas de vizinhança a serem usadas. Inicialmente, as *nv* vizinhanças {V_1, V_2, ..., V_{nv}} são ordenadas de acordo com algum critério (linha 1). Os vizinhos da solução corrente *S* são gerados, inicialmente, de acordo com a primeira estrutura de vizinhança (k=1) e o melhor vizinho é selecionado e atribuído à variável *S'* (linha 3). Se *S'* é melhor que a solução corrente (linha 4), *S'* substitui *S* e o processo é reiniciado (linha 6 e 8). Caso contrário (linhas 9 e 10), troca-se a estrutura de vizinhança e explora-se a próxima da lista *V* (V_{k+1}). O algoritmo é encerrado quando todas as vizinhanças tiverem sido exploradas e nenhuma resultar em uma solução *S'* melhor. Porém, sempre que uma solução *S'* de melhor qualidade é encontrada, o reínicio é feito a partir da primeira vizinhança (V_1).

A diferença entre o RVND e o VND é que o primeiro não obedece a ordenação definida para as estruturas de vinhança. Nele, as estruturas de vizinhanças são arranjadas de forma aleatória e a cada iteração uma vizinhança é escolhida aleatoriamente. Esta estratégia já foi utilizada por Subramanian *et al.* (2010). Porém, este trabalho propõe o uso de uma probabilidade de seleção para cada vizinhança. Esta probabilidade é proporcional à frequência de melhorias obtidas com a respectiva vizinhança. Assim, as melhores vizinhanças terão maior probabilidade de serem escolhidas. A seleção da vizinhança é feita através do método conhecido como *Roulette Wheel* (roleta), onde uma vizinhança V_i é sorteada baseado em sua probabilidade *P*.

O Algoritmo 3 apresenta o pseudocódigo do método RVND. Cada estrutura de vizinhança *k* inicia com a mesma probabilidade, dada por 1/*nv*, onde *nv* corresponde a quantidade de estruturas de vizinhança utilizadas (linha 2). A probabilidade de cada estrutura

de vizinhança é armazenada na lista *P*, em ordem compatível com a da lista de vizinhanças *V*. A cada iteração, a probabilidade P_k aumenta caso a vizinhança *k* melhore a solução (linha 10).

```
Algoritmo 3: RVND( S, nv )
   Entrada:
      S: Solução que deve ser melhorada
      nv: número máximo de estruturas de vizinhanças
   Início
 1    V ← {V1,V2,...,Vnv}; //Conjunto de estrutura de vizinhança
 2    P ← InicializaProbabilidade();//Inicia lista de Probabilidade P
 3    n ← 1; //Controla o número de vizinhanças sem melhora
 4    L ← ∅; //Lista de soluções proíbidas
 5   Enquanto (n ≤ nv) faça
 6     k ← EscolherVizinhanca(P, V-L);
 7     S' ← Melhor_Vizinho(Vk, S);
 8     Se ( f(S') < f(S) ) Então
 9       S ← S';
10       Pk ← AtualizarProbabilidade(Vk);
11       L ← ∅; //todas as vizinhanças poderão ser escolhidas
12       n ← 1; //reinício
13     Senão
14       L ← L ∪ {Vk}; //Vk não será escolhida na próxima iteração
15       n ← n + 1;
16   Fim_Enquanto
17   Retorna S;
   Fim
```

Além dessas estruturas, também são inicializadas uma variável *n* (linha 3), que armazena o número de estruturas de vizinhança exploradas mas que não obtiveram sucesso em melhorar a solução corrente e uma lista *L* de vizinhanças proibidas para escolha (linha 4), ou seja, a lista *L* é usada para armazenar as vizinhanças que não produziram melhoria da solução corrente *S* (linha 14). Dessa forma, as vizinhanças em *L* são proibidas de serem escolhidas pelo método "*EscolherVizinhanca*", que recebe como parâmetros a lista de probabilidade *P* e as vizanhanças disponíveis para serem escolhidas ($V - L$), e devolve o número da estrutura de vizinhança que será explorada (linha 6). A estrutura de vizinhança selecionada é passada como parâmetro para o método "*Melhor_Vizinho*", juntamente com a solução corrente S (linha 7). O método "*Melhor_Vizinho*" gera toda a vizinhança de *S*, de acordo com a estrutura V_k, e retorna a solução vizinha de melhor qualidade. Se a solução *S'* retornada é melhor que a solução atual *S* (linha 8), é feito a atualização de *S* (linha 9), a probabilidade de escolha de V_k é aumentada e todas as estruturas de vizinhanças são liberadas (linha 11), juntamente com o reínicio do algoritmo (linha 12).

Em suma, o algoritmo consiste em realizar iterações até que todas as estruturas de vizinhanças sejam exploradas e nenhuma melhora ocorra (linha 5), ou seja, $L=V$. A cada

iteração uma estrutura de vizinhança é escolhida (linha 6) e o melhor vizinho gerado por ela é selecionado (linha 7). Caso ele seja melhor que a solução atual, ocorre a atualização de *S* (linha 9) e a atualização da probabilidade associada a estrutura de vizinhança (linha 10). Em seguida, o reínicio ocorre (linhas 11 e 12).

As estruturas de vizinhanças propostas para este algoritmo são:

- ***V₁* – Operador de Inserção**: Para uma dada sequência $S=\{j_1,..,j_{x-1},j_x,j_{x+1},...,j_{y-1},j_y,j_{y+1},...,j_n\}$, este operador gera uma solução vizinha inserindo uma tarefa j_x (que está na posição x) em outra posição y tal que $y \neq x$ e $y \neq (x-1)$. Então, executando este movimento obtém-se: $S'=\{j_1,...,j_{x-1},j_{x+1},...,j_{y-1},j_x,j_y,j_{y+1},...,j_n\}$. A exploração desta vizinhança gera $(n-1)^2$ vizinhos.
- ***V₂* – Troca entre 2 tarefas**: Para uma dada sequência $S=\{j1,...,j_{x-1},j_x,j_{x+1},...,j_{y-1},j_y,j_{y+1},...,j_n\}$, este operador gera uma solução vizinha trocando as posições de duas tarefas diferentes j_x e j_y, ou seja, obtém-se $S'=\{j_1,..,j_{x-1},j_y,j_{x+1},...,j_{y-1},j_x,j_{y+1},...,j_n\}$. A exploração desta vizinhança gera $\frac{n(n-1)}{2}$ vizinhos.
- ***V₃* – Troca entre 2 tarefas sucessivas**: Dada uma sequência $S=\{j_1,...,j_{x-1},j_x,j_{x+1},...,j_{y-1},j_y,j_{y+1},...,j_n\}$, o movimento de troca entre duas tarefas sucessivas consiste em trocar j_{x-1} e j_x, respectivamente, com j_{y-1} e j_y, assim obtém-se uma nova sequência $S'=\{j_1,...,j_{y-1},j_y,j_{x+1},...,j_{x-1},j_x,j_{y+1},...,j_n\}$. A exploração desta vizinhança gera $\frac{(n-3)(n-2)}{2}$ vizinhos.

Além dessas estruturas de vizinhanças também foi avaliada uma quarta que ao final foi desprezada, por não apresentar grandes melhorias na qualidade da solução e consumir um elevado tempo de CPU, conforme mostrado em Campos *et al.* (2013). Ainda em Campos et al. (2013), é possível ver uma análise sobre a qualidade das vizinhanças e, também, a comparação do GRASP-RVND com um algoritmo da literatura.

2.3.2 ILS (Iterated Local Search)

O algoritmo ILS é uma metaheurística simples e robusta, que de acordo com Lopes *et al.* (2013) é formada por quatro operadores: Construção da solução inicial, busca local, pertubação e critério de aceitação.

Assim, como a maioria das metaheurísticas, o ILS inicia a partir de uma solução inicial. Após a geração da solução inicial, o ILS aplica iterativamente a busca local na solução corrente, explorando assim um dado ótimo local. Para que o algoritmo não fique estagnado em um ótimo local, é aplicado o operador de perturbação, que muda a solução corrente a fim de promover a exploração de outra área do espaço de soluções. Em seguida, aplica-se o operador de aceitação que define se a solução corrente será considerada para próxima iteração ou não. Uma solução melhor sempre será aceita; porém, uma solução de pior qualidade é aceita através de um fator probabilístico.

O Algoritmo 4 mostra o pseudocódigo do ILS. A idéia central deste algoritmo é ir aumentando o nível (tamanho) da pertubação, à medida que o algoritmo não consegue encontrar melhores soluções, possibilitando assim uma fuga mais rápida de um ótimo local.

Algoritmo 4: ILS(*CritérioParada, n, T, dmax, β*)

```
Entradas:
 CritérioParada: Critério de parada do algoritmo;
 n: Número de tarefas do problema;
 T: Tamanho inicial da perturbação;
 dmax: Condição para aumento do tamanho da perturbação;
 β: Probabilidade de aceitar uma solução pior;
Inicio
 1  S ← GeraSoluçãoInicial( ); //FRB4*1
 2  S ← BuscaLocal(S);
 3  S* ← S;
 4  numPert ← T;
 5  d ← 0;
 6  Enquanto ( Critério_Parada não satisfeito ) Faça
 7    S' ← Perturbação(S, numPert);
 8    S'' ← BuscaLocal(S', S*);  //RLS
 9    Se ( f(S'') < f(S*) ) então
10      S* ← S'';
11      numPert ← T;
12      d ← 0;
13    Senão
14      d ← d + 1;
15      Se ( d ≥ dmax ) Então
16        numPert ← numPert + T;
17        Se (numPert > n) Então
18            numPert = T;
19        d ← 0;
20    Se ( f(S'') < f(S) ) então
21      S ← S'';
22    Senão
23      Se ( rand(0..1) ≤ β ) então
24        S ← S'';
25  Fim_Enquanto
26  Retorna S*;
 Fim
```

O ILS recebe como entradas o critério de parada (parâmetro *CritérioParada*), que neste trabalho é utilizado tempo de CPU, o tamanho (número de tarefas) da instância do problema (parâmetro *n*), o nível (tamanho) inicial do processo de pertubação (parâmetro *T*), a condição para aumentar o nível de pertubação de uma solução (parâmetro d_{max}) e, por fim, a probabilidade de aceitação da solução (parâmetro β).

O algoritmo inicia gerando uma solução inicial *S* através do método "*GeraSoluçãoInicial*" e, em seguida, é aplicado o processo de busca local (linhas 1 e 2). A variável *S** armazena a melhor solução e é retornada ao final do algoritmo; na linha 3, ela recebe a solução inicial gerada. Nas linhas 4 e 5 as variáveis *numPert*, responsável por controlar o nível do processo de pertubação, e a variável *d*, responsável por armazenar o número de iterações sucessivas sem que haja melhoria na qualidade da solução *S**, são inicializadas.

A partir da linha 6, o *looping* principal do algoritmo é iniciado e uma pertubação na solução corrente *S* (linha 7) é realizada, produzindo uma nova solução (nova sequência) chamada de *S'*. Sobre esta solução é aplicado o processo de busca local (linha 8), produzindo então uma nova solução, chamada de *S''*. Os processos de pertubação e busca local são explicados com mais detalhe na próxima subseção.

Após estes passos, é feito o teste de aceitação da nova solução *S''* (linha 9). Se ela é melhor que a melhor solução obtida até o momento (*S**), então a atualização de *S** ocorre (linha 10). Além disso, as variáveis *numPert* e *T* são reinicializadas (linhas 11 e 12). Caso contrário, as variáveis *d* e *numPert* serão atualizadas, sendo que a primeira será incrementada em 1 (linha 14), e a segunda, será aumenta em *T* vezes (linha 16), caso seu valor exceda ao valor de d_{max} (linha 15), o que significa que o número de iterações sem melhora foi extrapolado. O aumento de *numPert* produz pertubações de tamanhos (nível) maiores. Percebe-se que, se o tamanho da pertubação exceder o número de tarefas do problema, ele é reajustado novamente para o valor *T* (linha 17 e 18).

Também é avaliado se o *S''* é melhor que a solução corrente *S* (linha 20). Caso seja verdadeiro, é realizado a atualização de *S* e, na próxima iteração, a pertubação e a busca local serão aplicadas sobre ela. Porém caso o teste não seja verdadeiro, há uma probabilidade β de considerar *S''* para próxima iteração (linhas 23 e 24).

2.3.2.1 Geração da Solução Inicial

A geração da solução inicial é um fator importante para o sucesso de uma metaheurística (TALBI, 2009). Quanto melhor a solução inicial, mais rapidamente a metaheurística tende a convergir para um ótimo local. Por este fato, testamos duas heurísticas com comprovado histórico de sucesso na resolução de problemas de programação da produção: NEH (NAWAZ et al, 1983) e $FRB4^*_1$ (PAN e RUIZ, 2014).

NEH é uma heurística construtiva gulosa, tradicional da literatura, que funciona inserindo e testando cada tarefa t_j de uma sequência candidata T (t_1, t_2, ..., t_n), ordenada por um dado critério, em todas as posições de uma sequência parcial S. A tarefa t_j deve permanecer na posição em que se obtém o melhor resultado para $f(S)$. O procedimento se repete até que todas as tarefas de T tenham sido inseridas na sequência S, formando assim uma sequência completa.

O algoritmo $FRB4^*_1$ é uma variação do algoritmo $FRB4_k$ proposto por Rad et al. (2009) e é apresentado no Algoritmo 2. A ideia do $FRB4_k$ é similar ao NEH; porém, após encontrar a melhor posição p para inserção de uma dada tarefa corrente t_j na sequência parcial S, exploram-se também k tarefas antecessoras (s_{j-1}, s_{j-2}, ..., s_{j-k}) e sucessoras (s_{j+1}, s_{j+2}, ..., s_{j+k}) em relação a tal posição p. Assim, as k tarefas antecessoras e k tarefas sucessoras são reinseridas iterativamente na sequência parcial S, testando todas as posições até encontrar a posição que resulta no melhor valor da função objetivo. A rotina segue em repetição até que todas as tarefas antecessoras e sucessoras tenham sido reinseridas na sequencia S. Ao final, a sequência de melhor qualidade é retornada. Neste trabalho, utiliza-se k=1 e é por isso que seu nome é denotado por $FRB4^*_1$.

Pan e Ruiz (2014) aperfeiçoaram o $FRB4_k$ propondo que ele seja aplicado sobre uma lista candidata parcial de tamanho λ, criada de acordo com alguma regra de despacho, por exemplo, EDD (*Earliest Due Dates*) para casos de minimização de atrasos ou SPT (*Shortest Processing Time*) para casos de minimização de tempos de processamento, sendo λ um parâmetro do algoritmo. Os benefícios decorrentes desta mudança são dois: o primeiro é que o tempo de CPU e complexidade computacional são reduzidos, visto que serão exploradas em um número menor de sequências parciais; o segundo, é que a ordenação criada pela aplicação da regra de despacho não é totalmente destruída, isto é, parte da qualidade obtida é mantida. Em certo casos, conforme demonstrado por Pan e Wang (2012), a destruição e reconstrução de toda a sequência geram resultados piores aos que foram alcançados inicialmente com a aplicação da regra de despacho.

O algoritmo 5 é uma versão adaptada ao português do FRB4*$_1$ de Pan e Ruiz (2014).

```
Algoritmo 5: FRB4*1( λ )
    Entradas:
     λ: tamanho da sequência parcial gerada inicialmente;
    Inicio
  1  //Ordenar as tarefas de acordo a regra de despacho
  2  β ← OrdenarTarefas(Regra);
  3  //Definir a sequência parcial π gerada a partir de β
  4  π ← {β0, β1, β2, ..., βλ-1};
  5  Para ( l ← λ até n ) Faça
  6    //Testar a inserção da tarefa β1 em todas as posição de π
  7    p ← MelhorPosicao(β1 , π);
  8    //Inserir a tarefa β1 na posição p da sequência π
  9    InserirTarefa(β1 , p, π);
 10    Para ( m ← max(1, p-1) até min(l, p+1) ) Faça
 11      //Extrair a tarefa t da posição m da sequência π
 12      t ← ExtrairTarefa(π, m);
 13      p ← MelhorPosicao(t , π);
 14      InserirTarefa(t , p, π);
 15    Fim_Para
 16  Fim_Para
 17  Retorna π
    Fim
```

Neste trabalho, o parametro λ foi configurado conforme o proposto pelos autores, isto é, $\lambda = {}^{n}/_{2}$. O que significa que 50% da sequência ordenada pela regra de despacho será mantida antes das inserções feitas pelo algoritmo. Foram feitos experimentos computacionais para definir a melhor regra de despacho para o problema e eles serão apresentados na seção de "Experimentos Computacionais".

2.3.2.2 Busca Local RLS

O procedimento da busca local é feito através do método RLS (*Referenced Local Search*) proposto por Pan *et al.* (2008) e aplicado por diversos autores em diversos tipos de problema de programação da produção, entre eles, os trabalhos de Ruiz (2010), Pan e Ruiz (2012) e Pan e Ruiz (2014). A RLS é uma busca local baseada na exploração de vizinhos por meio do operador de inserção, porém diferentemente de outros métodos de busca local por inserção, aqui a busca é conduzida por uma outra solução de melhor qualidade, chamada de "solução referência".

O pseudocódigo do RLS é apresentado no Algoritmo 6.

```
Algoritmo 6: RLS(S, S*, n)
  Entradas:
    S: Sequência onde será aplicada a busca local;
    S*: Sequência de referência para escolha das tarefas em S;
    n: Tamanho da sequência;
  Início
 1  cont ← 0;   i ← 0;
 2  Faça
 3    p ← Localizar(S,S*[i]); //retorna a posição de S*[i] em S
 4    S' ← S - S[p];
 5    //Inserir S*[i] na posição p que resulta no menor valor de f(S')
 6    S' ← InserirNaMelhorPosicao(S',S*[i]); //Retorna melhor sequência
 7    Se ( f(S') < f(S) ) Então
 8       S ← S';
 9       cont ← 1; //reinício
10     Senão
11       cont ← cont + 1;
12    i ← (i + 1 mod n);
13  Enquanto( cont < n );
14  Retorna S;
  Fim
```

O algoritmo recebe como entrada uma sequência *S*, correspondente à solução corrente, onde deve ser aplicada a busca local, e uma sequência *S**, que é a melhor solução conhecida ("solução referência"), conforme apresentado no Algoritmo 6. Em seguida, são feitas a remoção e reinserção das tarefas da sequência *S*, de forma iterativa, de acordo com o NEH, até que todas as tarefas tenham sido movimentadas e nenhuma melhora tenha sido gerada. No entanto, a definição sobre qual tarefa será movimentada é feita de acordo com a ordem das tarefas na sequência *S**. Ou seja, a primeira tarefa a ser desconectada e reinserida na sequência *S* é a que se encontra na primeira posição de *S** ($S^*_{[0]}$), a segunda tarefa será a $S^*_{[1]}$, e assim sucessivamente ($S^*_{[i]}$, $S_{*[i+1]}$, ..., $S^*_{[n]}$). O processo de exploração da solução *S* reinicia sempre que uma melhora em *f*(*S*) é encontrada e o algoritmo continua com a tarefa $S^*_{[i+1 \bmod n]}$.

2.3.2.3 Perturbação

A perturbação é uma estratégia para fazer com que a busca local explore outras áreas do espaço de solução e não fique presa em apenas um único ótimo local. Ela deve ser grande o suficiente para fazer com que outro ótimo local seja explorado, mas, ao mesmo tempo, não pode ser tão grande ao ponto de perder boas características já encontradas na solução, fazendo assim com que a metaheurística se torne um procedimento de busca aleatória.

Neste trabalho, o procedimento de perturbação é aplicado de forma semelhante ao trabalho de Rego *et al.* (2012), e em níveis, como proposto por Jacob *et al.* (2013). Ou seja, após um número d_{max} de iterações sem que haja melhora da solução *S**, o tamanho da

perturbação (nível) é aumentado. A ideia é garantir que a busca local faça a exploração apropriada dos vizinhos próximos à solução corrente, antes de se deslocar para outro local mais distante do espaço de soluções.

O procedimento de perturbação funciona fazendo a inversão de um bloco de tarefas, com tamanho *numPert*, através de um operador de troca de tarefas. A escolha do bloco é aleatória e as tarefas são trocadas de acordo com o nível da perturbação. Por exemplo, para o nível k, o valor de *numPert* é $k \times 2$ (significa que $k \times 2$ tarefas serão movimentadas), então as tarefas $j_1, j_2, ..., j_k$ são trocadas, respectivamente, pelas tarefas $j_{k\times2}, j_{k\times2-1},..., j_{k+1}$.

A Figura 3 (a) e (b) ilustram o movimento de perturbação produzido, respectivamente, para os níveis 2 e 3. Nota-se, através da Figura 3(b), que quando a posição de troca é maior que n, o movimento é realizado com as tarefas das primeiras posições.

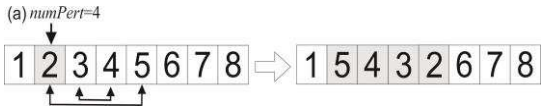

Figura 3 (a): Perturbação de nível 2 (numPert=4) realizada na sequência 12345678 com início na tarefa 2

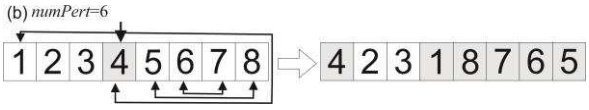

Figura 3 (b): Perturbação de nível 3 (numPert=6) realizada na sequência 12345678 com início na tarefa 4

2.3.3 IG (Iterated Greedy)

O IG é um método simples e eficaz proposto por Ruiz e Stützle (2007) e, posteriormente, utilizado na literatura por diversos outros autores. O algoritmo inicia com a construção de uma solução inicial e, em seguida, aplica iterativamente (até que um critério de parada seja satisfeito) seus dois operadores básicos: destruição-construção e busca local.

O operador de destruição-construção (DC) remove k tarefas de uma sequência S, e, em seguida, faz a reinserção de cada uma delas ao gosto da heurística NEH. Nós avaliamos duas formas para o operador de destruição-construção: a primeira, tradicional (DCa), que remove k tarefas aleatórias para reinserção, e, uma segunda (DC*), que remove um bloco de k tarefas adjacentes, ou seja, apenas a posição p ($p \leq n–k$) de início do bloco é definida aleatoriamente.

O Algoritmo 7 apresenta o pseudocódigo do IG. Ele recebe como parâmetros de entrada o critério de parada, que, neste trabalho, é o tempo de CPU, a quantidade de tarefas que serão desconectadas da sequência (parâmetro k) e a probabilidade de aceitação de uma solução pior (parâmetro β, onde $0 \leq \beta \leq 1$). O início do algoritmo ocorre a partir da geração de uma solução inicial (linha 1), utilizando a heurística $FRB4^*_1$ apresentada na seção 2.3.2.1. É aplicada uma

busca local na solução inicial gerada (linha 2). Neste algoritmo, também foi utilizada a busca local RLS apresentada na seção 2.3.2.2. Na linha 3 é feito o armazenamento da melhor solução conhecida, usando a variável *S**.

No *looping* principal do algoritmo, o operador DC é aplicado sobre a solução corrente *S* (linha 5), gerando uma nova sequência de tarefas (solução) *S'*, que, por sua vez, será submetida a busca local RLS (linha 6), gerando uma outra solução denominada *S''*. Note que a busca RLS é guiada pela solução *S**. Após a realização da busca local é feito o processo de aceitação da solução *S''*. Neste processo, duas verificações são realizadas para:

- **Atribuição de *S''* em *S***: A solução *S* é a solução que recebe a pertubação provocada pelo operador DC. Caso *S''* seja melhor que *S*, será então feita a atualização de *S* (linhas 7 e 8). Caso contrário, existe uma probabilidade de aceitar *S''* (linhas 11 e 12), controlada pelo parâmetro β.
- **Atribuição de *S''* em *S****: *S** corresponde a melhor solução conhecida pelo algoritmo, neste caso, se a qualidade de *S''* é maior que a atual *S**, a atribuição será realizada (linhas 9 e 10).

Após os testes de aceitação e atualização das variáveis, o algoritmo continua até que o critério de parada seja satisfeito.

```
Algoritmo 7: IG (CritérioParada, K, β)
  Entradas
    K: número de tarefas desconectadas de S na rotina DC;
    β: Probabilidade de aceitar uma solução pior;
  Inicio
 1  S ← GeraSoluçãoInicial( ); //FRB4*1
 2  S ← BuscaLocal(S,S); //RLS
 3  S* ← S;
 4  Enquanto ( Critério_Parada não satisfeito ) Faça
 5    S' ← DC(S, k); //Aplica DCa ou DC*;
 6    S'' ← BuscaLocal(S', S*); //RLS
 7    Se ( f(S'') < f(S) ) então
 8      S ← S'';
 9      Se ( S'' < S* ) Então
10        S* ← S'';
11    Senão ( rand(0..1) ≤ β ) então
12      S ← S'';
13  Fim_Enquanto
14  Retorna S*;
  Fim
```

2.3.4 ILS-IG

A ideia central do algoritmo ILS-IG é combinar a aplicação do ILS com o IG. Percebe-se que o ILS tem grande poder de escapar de ótimos locais através do seu operador de pertubação com níveis, ao passo que o IG consegue convergir rapidamente para pontos mais inferiores de um ótimo local (considerando o critério de minimização). Assim, o ILS-IG é uma tentativa de combinar essas duas boas características.

O Algoritmo 8 apresenta o pseudocódigo do ILS-IG.

```
Algoritmo 8: ILS-IG(CritérioParada, T, dmax, k, β)
 Entradas:
   T: Tamanho inicial da perturbação;
   dmax: Condição para aumento do tamanho da perturbação;
   k: número de tarefas desconectadas de S na rotina DC;
   β: Probabilidade de aceitar uma solução pior;
 Inicio
 1  S ← GeraSoluçãoInicial( ); //FRB4*1
 2  S ← BuscaLocal(S, S); //RLS
 3  S* ← S;  numPert ← T;   d ← 0;
 4  Enquanto ( Critério_Parada não satisfeito ) Faça
 5    Melhora ← 0; //indica se uma solução melhor foi encontrada
 6    S' ← DC(S, k); //Aplica DCa ou DC*
 7    S'' ← BuscaLocal(S', S*); //RLS
 8    Se ( f(S'') < f(S) ) então
 9      S ← S'';
10      Se ( S'' < S* ) Então
11         S* ← S''; Melhora ← 1;
12    Senão ( rand(0..1) ≤ β ) então
13      S ← S'';
14    Se ( Melhora > 0 ) então
15      numPert ← T;
16      d ← 0;
17    Senão
18      d ← d +1;
19      Se ( d ≥ dmax ) Então
20         S ← Perturbação(S*, numPert);
21         numPert ← numPert + T;
22         Se (numPert > n) Então
23            numPert = T;
24         d ← 0;
25  Fim_Enquanto
26  Retorna S*;
 Fim
```

É possível perceber que o processo de busca local do ILS foi substituído pelos operadores de "Destruição-Construção" (DC) e busca local do IG (linhas 6 e 7). Dessa forma, o IG é aplicado, sucessivamente, até um número d_{max} de iterações sem melhora na solução corrente. Quando isso ocorre o operador de pertubação do ILS é acionado. Ou seja, um ótimo

local é exaustivamente explorado antes de se efetuar uma perturbação maior na solução, que faz com que uma outra área do espaço de solução seja explorada.

Os parâmetros de entrada são os mesmos recebidos pelos algoritmos ILS e IG. Assim como os anteriores, o ILS-IG também inícia a partir da geração de uma solução inicial através da heurística $FRB4^*_1$ e aplicação da busca local RLS (linhas 1 e 2). Na linha 3, a variável que armazena a melhor solução conhecida, *S**, e as variáveis de controle do tamanho da pertubação e do número de iterações sem melhora, respectivamente *numPert* e *d*, são inicializadas. Em seguida, o *looping* principal do algoritmo começa. É utilizado uma variável denomidada *Melhora* para marcar se a iteração resultou em uma solução de melhor qualidade. A cada iteração ela é inicializada com o valor zero (linha 5) e quando uma solução melhor é encontrada o valor 1 lhe é atribuído (linha 11). Tal qual no IG, o operador DC é aplicado na linha 6 resultando em uma solução *S'*. Na linha 7 a busca local RLS é aplicada sobre *S'* gerando uma nova solução *S''*. O intervalo iniciado na linha 8 e terminado na linha 13, corresponde ao processo de aceitação do IG. Na linha 14 é verificado se houve melhora na solução. Caso o teste seja verdadeiro, as variáveis de controle *numPert* e *d* são reinicializadas (linhas 15 e 16). Caso contrário, se não houver melhora, a variável *d* será incrementada, e, de acordo com o parâmetro d_{max} o processo de pertubação do ILS será aplicado (linhas 19 a 24). Ao final, a solução armazenada por *S** será retornada.

2.3.5 Simulated Annealing Híbrido – PSA

O algoritmo PSA de Allahverdi e Aydilek (2013), proposto para o 2sAFS com objetivo de minimizar o atraso total. De acordo com o referido trabalho, o PSA foi o algoritmo que apresentou os melhores resultados em comparação com outros onze algoritmos. Para abordagem deste trabalho, o PSA foi reimplementado para ser aplicado sobre o problema aqui tratado, ou seja, para minimização do atraso total em um ambinete *assembly flowshop* com três estágios e tempos de preparação dependentes da sequência.

O PSA é a combinação de dois algoritmos heurísticos: *simulated annealing* (SA) e um algoritmo baseado na realização do movimento de inserção de tarefas na sequência, denominado PIA (*Proposed Insertion Algorithm*). O PSA consiste, basicamente, na aplicação do SA sobre uma solução inicial gerada, e, após o término do procedimento do SA, a solução obtida é passado ao PIA que, por sua vez, após o seu processamento, retorna uma nova solução melhorada.

O PSA é mostrado no Algoritmo 9.

```
Algoritmo 9: PSA(Ti, Tf, cf, Nr)
  Entradas:
  Ti: Temperatura inicial do SA;
  Tf: Temperatura final do SA;
  cf: Fator de resfriamento do SA;
  Nr: Número de iterações realizadas a cada temperatura T pelo SA;
  Inicio
1   S ← GeraSoluçãoInicial( );
2   S' ← SA(Ti, Tf, cf, Nr, S); //aplicação do SA
3   S* ← PIA(S') //aplicação do PIA
4   Retorna S*;
  Fim
```

Os valores utilizados para os parâmetros *Ti*, *Tf* e *cf* são iguais aos de Allahverdi e Aydilek (2013) que são, respectivamente, 0.1, 0.0001, 0.98. Já o parâmetro *Nr* foi regulado de forma a proporcionar que o PSA rode em tempo igual ou próximo ao tempo de execução estipulado para os outros algoritmos.

A geração da solução inicial considerada pelo PSA é dada pela ordenação das tarefas (em ordem crescente) de acordo com a matriz unidimensional *s_SPTa*[1..*n*]. Tal matriz é gerada a partir da soma de outras duas matrizes: *M_t* e *m_t*. Onde *M_t* corresponde ao maior tempo de processamento, no estágio 1 para uma dada tarefa, enquanto, *m_t* é computado pelo menor tempo de processamento também no estágio 1. As equações abaixo demostram estas operações:

$$s\_SPTa_j = M\_t_j + m\_t_j \quad \forall j = 1, \ldots, n$$

$$M\_t_j = \max_{k=1..m} t_{k,j} \quad \forall j = 1, \ldots, n$$

$$m\_t_j = \min_{k=1..m} t_{k,j} \quad \forall j = 1, \ldots, n$$

O Algoritmo 10 e o Algoritmo 11 mostram, respectivamente, os algoritmos SA e PIA usados como métodos do PSA. Ambos foram extraídos do trabalho de Allahverdi e Aydilek (2013), adaptados e traduzidos para o português.

O operador "*movimentoDeTroca*" realiza a troca de duas tarefas escolhidas aleatoriamente. Já o operador "*probabilidade*" retorna verdadeiro, caso um número gerado aleatoriamente seja inferior ao valor resultante da expressão $exp(-d / T)$.

Algoritmo 10: SA(Ti, Tf, cf, Nr, Si)

```
Entradas:
  Ti: Temperatura inicial;
  Tf: Temperatura final;
  cf: Fator de resfriamento;
  Nr: Número de iterações realizadas em cada temperatura T;
  Si: Solução inicial;
Inicio
 1  S ← Si;
 2  T ← Ti;
 3  Enquanto ( T >= Tf ) Faça
 4    j ← 1;
 5    Enquanto ( j < Nr ) Faça
 6      St ← movimentoDeTroca( S );
 7      Se ( f(St) < f(S) ) então
 8        S ← St;
 9      Senão
10        d ← ( f(St) - f(S) ) / f(S);
11        Se ( probabilidade( exp( -d/T ) ) ) então
12          S ← St;
13      j ← j+1;
14    Fim_Enquanto
15    T ← T * cf;
16  Fim_Enquanto
17  Retorna S;
Fim
```

Algoritmo 11: PIA(Si , Rn)

```
Entradas:
  Si: Solução inicial;
Inicio
 1  Sb ← Si; Sr ← Si;
 2  r ← 1;
 3  Enquanto ( r <= Rn ) Faça
 4    j ← 1;
 5    Enquanto ( j <= n ) Faça //n = número de tarefas do problema
 6      p ← 1;
 7      Enquanto ( p <= n ) Faça
 8        St ← Insercao(j, p, Sr);
 9        Se ( f(St) < f(Sb) ) então
10          Sb ← St;
11        p ← p+1;
12      Fim_Enquanto
13      j ← j+1;
14    Fim_Enquanto
15    r ← r + 1;
16    Sr ← Sb;
17  Fim_Enquanto
18  θ ← Sb;
19  i ← 1;
20  Enquanto ( i <= n ) Faça
21    ϕ ← Troca(i, i+1, θ);
22    Se ( f(ϕ) < f(θ) ) então
23      θ ← ϕ;
24  Fim_Enquanto
25  Retorna θ;
Fim
```

O operador "*Inserção*" do PIA (linha 8) remove a tarefa da posição *j* e a insere na posição *p* da sequência *Sr*. Já o operador "*Troca*" (linha 21) faz a troca de tarefas entre as posições *i* e *i*+1, ou seja, movimenta as tarefas para posições adjacentes em relação as posições em que se encontram. Em seu trabalho, Allahverdi e Aydilek (2013), utilizaram o valor 12 para o parâmetro *Rn*; porém, neste trabalho, optou-se por efetuar a saída do laço de repetição (linha 3) quando a última iteração não produzir melhoria na solução *Sb*, evitando que seja gasto tempo de CPU desnecessariamente.

2.4 Experimentos Computacionais

O objetivo desta seção é apresentar os resultados computacionais referentes ao desempenho das heurísticas propostas para o problema do 3sAFS com tempos de preparação dependentes da sequência das tarefas. Com o intuíto de extrair a melhor configuração de cada algoritmo, primeiramente, foram realizados experimentos para configurar a geração da solução inicial e o procedimento de destruição-construção do IG, além da calibração de todos os parâmetros dos algoritmos. Ao final, também é apresentada a comparação entre as propostas heurísticas.

A seguir, os critérios básicos usados para realização dos experimentos, como a forma de geração das instâncias dos problemas de testes, o critério de parada estabelecido para os algoritmos, as métricas de avaliação de desempenho e a metodologia de análise estatística, são expostos.

Todos os algoritmos foram codificados em C++ e executados no mesmo ambiente computacional: computador com processador Intel® Core™ i5-3570 CPU @ 3.40GHz, memória de 16GB e sistema operacional Windows 7 Professional de 64 bits.

2.4.1 Geração de instâncias do problema

Para realização dos experimentos foram gerados dois grupos diferentes de instâncias do problema. Um grupo com instâncias de pequeno porte, destinadas à experimentos relacionados ao modelo matemático, e um grupo de instâncias de grande porte, para experimentos destinados à avaliação do desempenho dos algoritmos. Para ambos os grupos, foram produzidas diferentes instâncias do problema, com tamanhos variados, de acordo com Liefooghe et al. (2012).

Para as instâncias de pequeno porte, a variação da quantidade de tarefas é $n \in \{8, 10, 12, 14\}$ e a quantidade de máquinas paralelas no primeiro estágio é $m \in \{2, 4, 6\}$, totalizando 12 combinações diferentes. Para cada combinação $n \times m$ foram geradas 10 diferentes instâncias aleatórias, totalizando assim 120 problemas. Já para as instâncias de grande porte, a variação da quantidade de tarefas é $n \in \{30, 50, 80, 100, 200\}$ e a quantidade de máquinas paralelas no primeiro estágio é $m \in \{5, 10, 20\}$, totalizando 15 combinações diferentes. Para cada combinação $n \times m$ foram geradas 10 diferentes instâncias aleatórias, totalizando assim 150 problemas.

Os tempos de processamento das tarefas no primeiro estágio ($t_{k,j}$), no segundo estágio ($tt_{k,j}$) e no terceiro estágio ($ta_{k,j}$) são inteiros, gerados aleatoriamente, a partir de uma distribuição uniforme no intervalo [0,99]. Os tempos de preparação de todos os estágios também são inteiros, gerados aleatoriamente, em uma distribuição uniforme no intervalo [0,49]. As datas de entrega (d_j) de cada tarefa foram geradas aleatoriamente, com uma distribuição uniforme no intervalo [3 × *p*, (*n*+2) × *p*], onde *p* é o valor médio dos tempos de processamento gerados anteriormente. Neste intervalo, o valor 3 é usado, porque o 3sAFS pode ser considerado como um *flow shop scheduling* com três máquinas. Assim, as datas de entrega geradas estão entre a média do tempo de conclusão da primeira tarefa da sequência e o tempo médio de conclusão da última tarefa da sequência (LIEFOOGHE *ET AL.*,2012).

2.4.2 Critério de parada dos algoritmos e métrica de avaliação

Como critério de parada dos algoritmos foi utilizado o tempo máximo de execução, que é definido pela fórmula: $n \times m \times c$, onde *n* corresponde ao número de tarefas, *m* corresponde ao número de máquinas no primeiro estágio e *c* é uma constante de tempo (em milissegundos). Para os experimentos de calibração foi usado *c*=50, já para o experimento de comparação dos algoritmos foi utilizado *c*=100.

A métrica de avaliação da qualidade das soluções obtidas por cada algoritmo é o Desvio Percentual Relativo (RPD em inglês, *Relative Percentage Derivation*), comumente utilizada em abordagens do problema de sequenciamento de tarefas (VALLADA ET AL, 2008). Sua fórmula de cálculo é apresentada a seguir, onde $f_{algoritmo}$ corresponde ao valor da função objetivo encontrado pelo algoritmo em questão e f_{melhor} é o melhor valor conhecido com base na execução de todos os algoritmos:

$$RPD = \frac{f_{algoritmo} - f_{melhor}}{f_{melhor}} \times 100$$

A utilização do RPD se justifica por não ter sido encontrado, no conjunto de instâncias de teste, nenhum resultado de atraso total com valor zero, em qualquer rodada dos algoritmos. Esta medida é considerada como a variável de resposta em todas as análises estatísticas realizadas para todos os experimentos computacionais.

2.4.3 Análise estatística

Para validar os experimentos foram utilizados métodos estatísticos para comprovar a existência de diferenças significativas entre os objetos analisados. Em todas as análises, foi utilizado, como variável de resposta, o RPD, calculado em cada rodada de cada algoritmo sobre cada uma das instâncias.

A metodologia de avaliação estatística, definida para este trabalho tem por princípio fazer primeiro a tentativa do uso da Análise de Variância (ANOVA) paramétrica. A ANOVA é um teste estatístico que visa, fundamentalmente, medir se há diferenças significantes entre médias. Porém, para sua utilização é necessário que os dados avaliados tenham caracterísiticas específicas, que são chamadas de "pressuposições". As pressuposições ANOVA são: igualdade de variância (homocedasticidade), normalidade e independência dos resíduos. Se todas as pressuposições forem verdadeiras, então a análise através da ANOVA continua; porém, caso contrário, se alguma pressuposição falhar, o uso da ANOVA se torna inválido.

Para os casos onde a ANOVA não pode ser aplicada, adota-se o critério de utilizar o teste não paramétrico *Kruskal-Wallis*, que é uma alternativa não-paramétrica, já conhecida na literatura, para casos onde a ANOVA paramétrica não pode ser aplicada. Porém, o teste de *Kruskal-Wallis* não apresenta quais médias são estatisticamente diferentes. Então aplica-se também o teste de múltiplas comparações, que compara cada par de médias com nível de confiança de 95% e mostra quais objetos analisados possuem diferenças significativas.

O teste de múltiplas comparações, em alguns casos, pode ser muito extenso e dificultar a análise e a visualização das diferenças entre as médias. Por este motivo, sempre ao final da aplicação dos testes, é apresentado um gráfico de intervalo e médias, resultante da aplicação do teste *Tukey* da Diferença Honestamente Significativa (HSD), com nível de confiança de 95%. Além de facilitar a visualização dos resultados, o gráfico resultante do teste de *Tukey* valida o

teste de múliplas comparações, pois, quando um intervalo de um objeto analisado no gráfico não é sobreposto aos intervalos de outros objetos, conclui-se que há diferenças significativas entre eles.

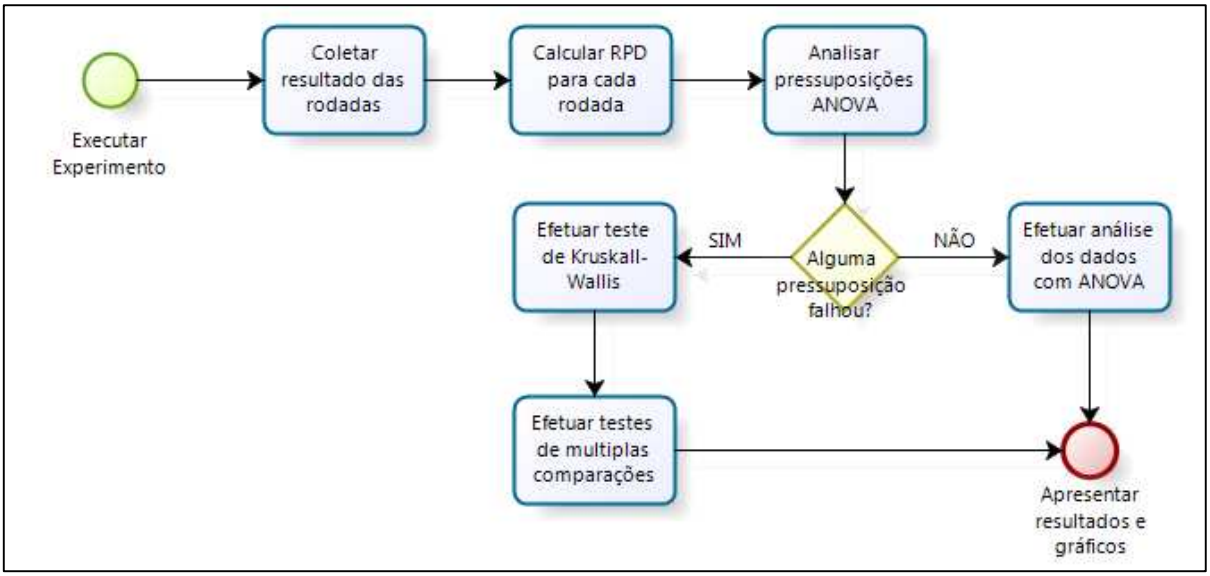

Figura 4: Método de avaliação dos resultados através de análises estatistícas

A Figura 4 apresenta, em formato de fluxograma, o modelo metodológico de avaliação de resultados adotado neste trabalho, mostrando a sequência de passos utilizada durante a aplicação dos testes estatísicos.

É importante ressaltar que, a falha de qualquer uma das pressuposições ANOVA é suficiente para invalidar o experimento. Sendo assim, avaliam-se as pressuposições na seguinte ordem: homoscedasticidade, normalidade e independência dos resíduos. Para avaliação de tais pressuposições utilizam-se os respectivos métodos: teste de *Levene's*, teste de *Shapiro-Wilk W* e gráfico de plotagem dos resíduos (AGEEL, 2000). Se, algum destes testes demonstrar que pelo menos uma das pressuposições é rejeitada, conclui-se que a utilização da ANOVA é inválida, e, sendo assim, não há necessidade da aplicação dos outros testes.

Todos os testes estatísticos foram realizados com o auxílio da ferramenta de apoio estatístico Statgraphics Centurion XVI[1].

[1] Informações detalhadas sobre a ferramenta Statisgraphics estão disponíveis em: http://info.statgraphics.com/statgraphics-home

2.4.4 Experimentos computacionais para configuração dos algoritmos

Para determinar a melhor configuração dos algoritmos, foram realizados experimentos para avaliar o comportamento de algumas partes dos algoritmos como: geração da solução inicial, busca-local e procedimento de destruição-construção do IG. Para isso, os algoritmos foram rodados com variações dessas partes e os resultados foram comparados.

Além destes testes, é importante definir os melhores valores para os parâmetros dos algoritmos. Assim, um experimento computacional de calibração é realizado, baseado na metodologia Desenho de Experimentos (DOE) (Montgomery, 2006), onde cada fator é um parâmetro controlado. Dessa forma, toda a faixa de valores disponíveis para cada parâmetro é combinada, gerando então um desenho fatorial completo, onde cada combinação é executada e comparada com as outras.

Estes experimentos foram realizados utilizando uma amostra de 50% do conjunto de instâncias de grande porte (75 instâncias). Para cada instância, os algoritmos foram executados cinco vezes em cada configuração de parâmetros. Para o critério de parada, a constante *c* foi definida com o valor 50, ou seja, o algoritmo para quando o tempo de execução atingir o valor dado pela fórmula: $n \times m \times 50$.

2.4.4.1 Experimento de construção da solução inicial do ILS, IG e ILS-IG

Para determinar a forma de geração da solução inicial, duas heurísticas diferentes foram avaliadas: NEH e FRB4$^*_1$. Neste experimento, foi utilizado o valor 3 para o parâmetro λ do FRB4$^*_1$. As duas heurísticas foram testadas usando diferentes regras (regras de despacho) para ordenação da lista candidata:

- ***EDD* (*Earliest Due Date*)**: Ordena as tarefas em ordem crescente de suas datas de entrega.
- ***TLB* (*Tardiness Lower Bound*)**: Ordena as tarefas em ordem decrescente do limitante inferior do atraso total, ou seja:

$$d_{[j]} - \left\{ \max_{k=1,\ldots,m} t_{[j,k]} + tt_{[j]} + ta_{[j]} \right\}, \forall_{j=1,\ldots,n}.$$

Dessa forma, o ILS foi executado com 4 versões diferentes da heurística de construção da solução inicial, dada pela combinação com as regras de despacho, denominadas: NEH–EDD, FRB4$^{*}_{1}$–EDD, NEH–TLB e FRB4$^{*}_{1}$–TLB.

Ao analisar os resultados, percebeu-se que a ANOVA não poderia ser aplicada, pois o teste *Levene's*, que avalia a pressuposição de homocedasticidade resultou em valor "*P-Value*" inferior a 0.05, conforme mostrado na Tabela 1 a seguir, indicando que há uma diferença significativa entre os desvios padrões.

Tabela 1: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de comparação da heurísticas de construção da solução inicial

	Test	*P-Value*
Levene's	80,7336	**0**

Sendo assim, o teste de *Kruskal-Wallis* foi aplicado e resultou em um valor "*P-Value*" inferior a 0.05, indicando que há diferenças estatisticamente significantes nas comparações realizadas. Para visualizar quais são as heurísticas estatisticamente diferentes foi aplicado o teste de Multiplas Comparações apresentado na tabela 2. A coluna "Diferença" mostra a média das amostras da primeira versão menos as da segunda. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer par de heurísticas comparadas, no qual o valor absoluto da diferença exceda o limite, indica que as versões são, de forma significativas, diferentes em termos estatísticos, com nível de confiança de 95%. Na tabela, a existência dessa diferença é indicado por um (*) na coluna "Significante".

Tabela 2: Resultado do teste de múltiplas comparações entre as heurísticas de construção de solução inicial

Comparação	Significante	Diferença	+/- Limite
FRB4*1-EDD - FRB4*1-TLB	*	**-0,180974**	0,0178654
FRB4*1-EDD - NEH-EDD	*	**-0,0572737**	0,0178654
FRB4*1-EDD - NEH-TLB	*	**-0,305004**	0,0178654
FRB4*1-TLB - NEH-EDD	*	**0,123701**	0,0178654
FRB4*1-TLB - NEH-TLB	*	**-0,12403**	0,0178654
NEH-EDD - NEH-TLB	*	**-0,247731**	0,0178654

Nota-se que todas as heurísticas apresentam diferenças significativas entre elas. Então, para compreender qual delas é capaz de gerar, em média, o melhor resultado para o problema, o gráfico de médias e intervalos resultante do teste de *Tukey* da Diferença Honestamente Significativa (HSD), com nível de confiança de 95%, é apresentado na Figura 5.

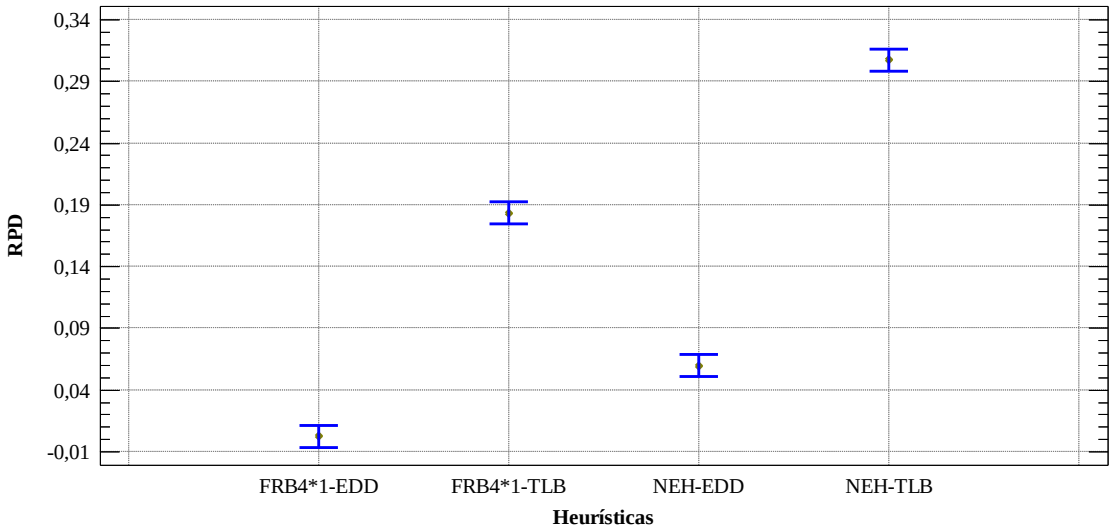

Figura 5: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para as heurísticas construtivas avaliadas

Nota-se, no gráfico da Figura 5, que o intervalo dos objetos analisados não estão sobrepostos; então, conclui-se que há diferença significativa entre suas médias. É perceptível que a heurística FRB4$^{*}_{1}$, ordenada com regra de despacho EDD, gerou melhores resultados.

2.4.4.2 Calibração do procedimento de destruição-construção do IG

O procedimento de destruição-construção do IG também foi testado. Aqui, foi comparada sua abordagem tradicional (DCa), que desconecta tarefas aleatoriamente, com uma abordagem de desconexão de tarefas por bloco (DC*). Assim o IG foi executado com essas duas diferentes configurações. Após a coleta dos resultados, o teste de *Levene's* foi aplicado (apresentado na Tabela 3) para avaliar a pressuposição de homoscedasticidade. O valor "*P-Value*" resultante ficou superior à 0.05, fato que confirma tal pressuposição.

Tabela 3: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA para comparação dos procedimentos DCa e DC*

	Test	*P-Value*
Levene's	0,232672	**0,629547**

A próxima pressuposição a ser verificada então é a normalidade. Então, o teste de *Shapiro-Wilk W* foi aplicado sobre os resíduos obtidos da distribuição dos dados. Porém como pode ser visto na Tabela 4 abaixo, o valor "*P-Value*" ficou inferior à 0.05, apontando que não há uma distribuição normal dos dados. Sendo assim, novamente a ANOVA deve ser desconsiderada e o teste não paramêtrico *Kruskal-Wallis* deve ser aplicado, juntamente com o teste de múltiplas comparações. O resultado "*P-value*" retornado pelo teste *Kruskal-Wallis* foi 0.104724, o que indica que não há diferenças significativas entre os métodos, pois é superior a

0.05. A Tabela 5 mostra a saída do teste de múltiplas comparações, apontando que não existem diferenças estatísticas entre eles.

Tabela 4: Teste *Shapiro-Wilk W* para avaliação da pressuposição de normalidade da ANOVA sobre os dados de comparação dos procedimentos DC(a) e DC*

Test	*Statistic*	*P-Value*
Shapiro-Wilk W	0,900642	**0,0**

Tabela 5: Resultado do teste de múltiplas comparações entre os algoritmos IG–DC(a) e IG–DC(*)

Comparação	**Sig.**	**Diferença**	**+/- Limite**
IG–DC(a) - IG–DC*		0,29778	0,317293

O teste de múltiplas comparações aponta que não há diferenças estatisticas significativas entre os procedimentos. Assim, é sugerido que a aplicação de qualquer um dos procedimentos não muda, de forma significativa, o resultado final (médio) do algoritmo. Porém, o gráfico de médias e intervalos (Figura 6) mostra que a utilização do DC*, em média, obtém resultados melhores que o DC(a). Assim, a opção pela utilização do DC* torna-se clara.

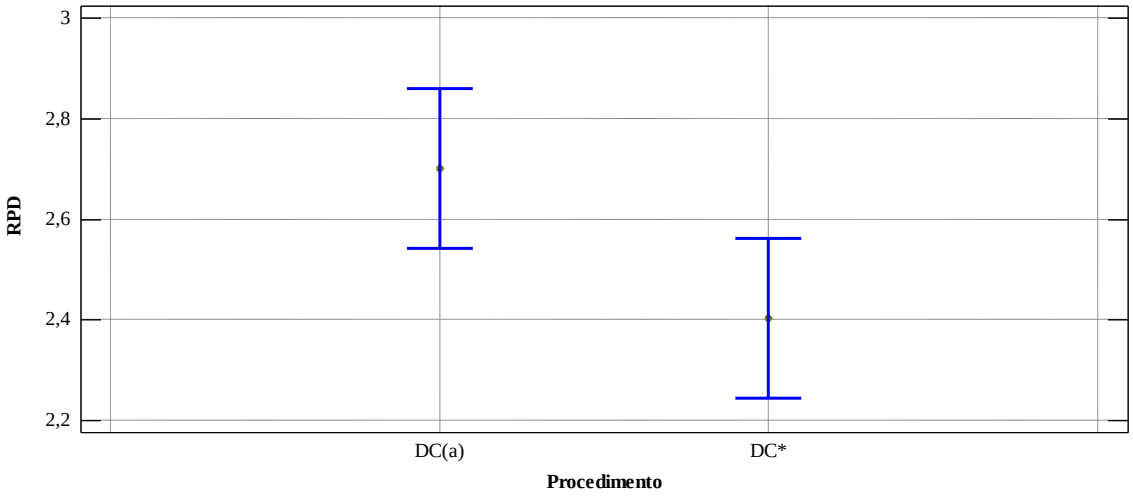

Figura 6: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação da execução do algoritmo IG com os procedimentos DC(a) e DC(*)

Através da Figura 6 pecebe-se que o procedimento DC*, em média, é melhor que o DC(a). Portanto, ele foi escolhido para compor os algoritmos IG e ILS-IG.

2.4.4.3 Calibração do parâmetro α do GRASP-RVND

O algoritmo GRASP-RVND possui apenas um parâmetro para ser configurado, que é o parâmetro α. Os seguintes valores foram testados: {0.1, 0.3, 0.5, 0.7, 0.9}. O teste de variância aplicado sobre os desvios percentuais relativos das rodadas mostra que os dados possuem homocedasticidade, validando a pressuposição ANOVA. Isso pode ser visto através da saída do teste de *Levene's* apresentado na Tabela 6. O valor da coluna "*P-Value*" ficou superior a 0.05, indicando que não há uma diferença entre os desvios padrões das médias.

Tabela 6: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de calibração de parâmetros do GRASP-RVND

	Test	*P-Value*
Levene's	11,3512	**4,1982E-9**

Com a pressuposição de homocedasticidade confirmada, a próxima pressuposição a ser verificada é a normalidade. Então, o teste de *Shapiro-Wilk W* foi aplicado sobre os resíduos obtidos da distribuição dos dados. Porém, como pode ser visto na Tabela 7, o valor "*P-Value*" ficou inferior à 0.05 (zero), apontando que não há uma distribuição normal dos dados. Isso então rejeita a pressuposição da normalidade e, assim, novamente a ANOVA é desconsiderada.

Tabela 7: Teste *Shapiro-Wilk W* para avaliação da pressuposição de normalidade da ANOVA sobre os dados de calibração de parâmetros do GRASP-RVND

	Test	*P-Value*
Shapiro-Wilk W	0,888057	**0,0**

Por este motivo utilizou-se o teste não paramétrico de *Kruskal-Wallis*. O valor "*P-value*" é inferior a 0.05 (*P-value*=0.0198069E-9), o que mostra que há diferenças significativas entre as configurações comparadas. A Tabela 8 mostra uma visualização alternativa a respeito do teste de múltiplas comparações. Nelas são apresentadas, de forma agrupada, as configurações que possuem médias homogêneas, ou seja, aquelas que não possuem diferenças estatísticas significantes entre si. Isso pode ser visto através da coluna "Grupos homogêneos": as configurações que possuem o "X" na mesma direção vertical pertencem ao mesmo grupo, ou seja, entre elas não há diferenças significativas. Também, é possível perceber quais são as menores médias para o desvio relativo padrão, atráves da coluna "RPD Médio", e a quantidade de amostras utilizadas no experimento, através da coluna "Amostra". Nota-se que a configuração com o parâmetro α=0.9 obteve a menor média e é, em termos estatísticos, diferente de todas as outras configurações.

Tabela 8: Apresentação do resultado médio da calibração do parâmetro α do algoritmo GRASP-RVND

Parâmetro α	**Amostra**	**RPD Médio**	**Grupos homogêneos**
0.9	375	2,40093	X
0.1	375	3,21542	X
0.3	375	3,27915	X
0.7	375	3,34665	XX
0.5	375	3,64197	X

A Tabela 9 mostra a saída padrão do teste de múltiplas comparações. A coluna "Diferença" mostra a média das amostras da primeira versão menos as da segunda. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer par de parâmetros comparados, no qual o valor absoluto da diferença exceda o limite, indica que as

versões são, de forma significativa, estatisticamente diferentes, com nível de confiança de 95%. Na tabela, a existência dessa diferença é indicada por um (*) na coluna “Significante”.

Tabela 9: Resultado do teste de múltiplas comparações entre as diferentes configurações do algoritmo GRASP-RVND

Comparação (α)	Significante	Diferença	+/- Limite
0.1 - 0.3		-0,0637223	0,360715
0.1 - 0.5	*	**-0,426545**	0,360715
0.1 - 0.7		-0,131224	0,360715
0.1 - 0.9	*	**0,814495**	0,360715
0.3 - 0.5	*	**-0,362823**	0,360715
0.3 - 0.7		-0,0675014	0,360715
0.3 - 0.9	*	**0,878217**	0,360715
0.5 - 0.7		0,295322	0,360715
0.5 - 0.9	*	**1,24104**	0,360715
0.7 - 0.9	*	**0,945719**	0,360715

É obeservado que a configuração com α=0.9 possui diferenças estatisticamente significantes para todas as outras configurações. Para validar o resultado apresentado, é gerado o gráfico de intervalo e médias na Figura 7. Nele são mostradas as médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95%. Quando o intervalo de um objeto analisado não sobrepõe aos outros intervalos, conclui-se que há diferença significativamente entre as médias. Pode-se observar que a configuração onde α=0.9 não é interceptada por nenhum outro intervalo. Além disso, seus limites superiores e inferiores são consideravelmente menores que as outras configurações. Este, então, será o valor usado para o parâmetro α do GRASP-RVND.

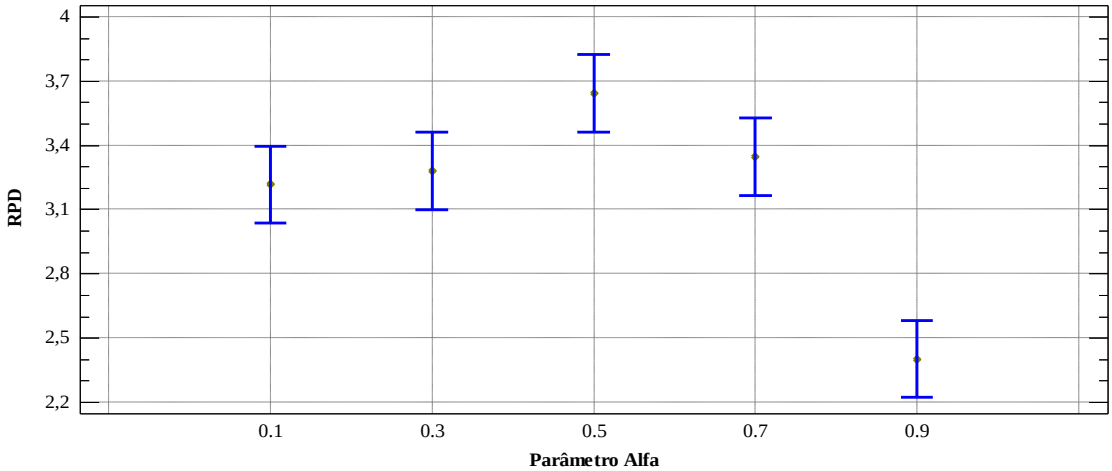

Figura 7: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para as configurações do parâmetro α do algoritmo GRASP-RVND

2.4.4.4 Calibração de parâmetros do ILS

Para o algoritmo ILS os seguintes parâmetros foram analisados:

- *T*: Tamanho inicial da perturbação;

- d_{max}: Número de iterações sem melhora realizada pelo algoritmo antes do nível da pertubação aumentar;
- β : corresponde à probabilidade de aceitação de uma solução pior.

Os seguintes conjuntos de valores foram adotados: $T \in \{2, 4\}$, $d_{max} \in \{15, 20, 25\}$ e $\beta \in \{0, 0.3, 0.6\}$. Dessa forma, foram testadas 18 configurações de parâmetros. A identificação destas configurações são mostradas na Tabela 10 abaixo.

Tabela 10: Combinação dos parâmetros T, d_{max} e β para o algoritmo ILS

Configuração	***T***	d_{max}	β	**Configuração**	**K**	d_{max}	**β**
1	2	15	0	10	4	15	0
2	2	15	0.3	11	4	15	0.3
3	2	15	0.6	12	4	15	0.6
4	2	20	0	13	4	20	0
5	2	20	0.3	14	4	20	0.3
6	2	20	0.6	15	4	20	0.6
7	2	25	0	16	4	25	0
8	2	25	0.3	17	4	25	0.3
9	2	25	0.6	18	4	25	0.6

O teste de variância, aplicado sobre os desvios percentuais relativos das rodadas mostra que a pressuposição de homocedasticidade da ANOVA é rejeitada, como pode ser visto no teste de *Levene's*, na Tabela 11. O valor da coluna "*P-Value*" ficou inferior a 0.05, indicando que há uma diferença significativa entre os desvios padrões, inviabilizando o uso da ANOVA.

Tabela 11: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de calibração dos parâmetros do ILS

	Test	*P-Value*
Levene's	5,66876	**0**

Por este motivo utiliza-se o teste não paramétrico de *Kruskal-Wallis*. O valor "*P-value*" resultante do teste é 0 (zero). Quando este valor ("*P-value*") é inferior a 0.05, indica que há uma diferença significativa entre as configurações testadas. Como o teste de *Kruskal-Wallis* não mostra quais são essas diferenças, é aplicado o teste de múltiplas comparações. As saídas deste teste são mostradas nas Tabelas 12 e 13.

A Tabela 12 mostra, de forma agrupada, as configurações que possuem médias homogêneas, ou seja, aquelas que não possuem diferenças estatísticas significantes entre si. Isso pode ser visto através da coluna "Grupos homogêneos": as configurações que possuem o "X" na mesma direção vertical pertencem ao mesmo grupo. Também é possível visualizar o resultado do RPD médio alcançado por cada configuração, através da coluna "RPD Médio", e a quantidade de amostras utilizadas no experimento (coluna "Amostras"). Observa-se que a configuração 8 possui a menor média e está no mesmo grupo das configurações 5, 4, 7 e 9.

Tabela 12: Apresentação dos grupos homogêneos das configurações de calibração dos parâmetros do algoritmo ILS

Configuração	Amostra	RPD Médio	Grupos homogêneos
8	375	3,35227	X
5	375	3,36837	X
4	375	3,64499	XX
7	375	3,70178	XXX
9	375	3,71928	XXXX
6	375	3,81681	XXXX
1	375	3,83909	XXXX
2	375	3,89676	XXXX
13	375	3,99595	XXXXX
3	375	4,04179	XXXXX
16	375	4,11554	XXXX
10	375	4,15517	XXX
17	375	4,31484	XX
14	375	4,39267	XX
11	375	4,74046	XX
18	375	4,81045	XX
15	375	4,87184	XX
12	375	5,18461	X

O mesmo resultado pode ser visto na Tabela 13, onde a saída padrão do teste de múltiplas comparações é apresentada. Nele, cada par de configuração é comparado e a coluna "Diferença" mostra a média das amostras da primeira versão menos as da segunda. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer par de versão, no qual o valor absoluto da diferença excede o limite, indica que as versões são, de forma significativa, estatisticamente diferentes, com nível de confiança de 95%. Na tabela, a existência dessa diferença é indicado por um (*) na coluna "Significante". É possível observar que assim como sugerido pela Tabela 12, as configurações 4, 5, 7, 8 e 9 não possuem diferenças estatísticas significantes entre si. Isso significa que se poderia utilizar qualquer uma delas que o resultado obtido pelo algoritmo não sofreria mudanças significativas.

Tabela 13: Resultado do teste de múltiplas comparações entre as diferentes configurações do algoritmo ILS

Comparação	*Significante*	*Diferença*	*+/- Limite*	*Comparação*	*Significante*	*Diferença*	*+/- Limite*
1 – 2		-0,0576743	0,396369	6 - 9		0,0975336	0,396369
1 – 3		-0,202701	0,396369	6 - 10		-0,338357	0,396369
1 – 4		0,194102	0,396369	6 - 11	*	**-0,923656**	0,396369
1 – 5	*	**0,470714**	0,396369	6 - 12	*	**-1,3678**	0,396369
1 – 6		0,0222791	0,396369	6 - 13		-0,179139	0,396369
1 – 7		0,137311	0,396369	6 - 14	*	**-0,575865**	0,396369
1 – 8	*	**0,486823**	0,396369	6 - 15	*	**-1,05503**	0,396369
1 – 9		0,119813	0,396369	6 - 16		-0,298732	0,396369
1 - 10		-0,316078	0,396369	6 - 17	*	**-0,498028**	0,396369
1 - 11	*	**-0,901377**	0,396369	6 - 18	*	**-0,993646**	0,396369
1 - 12	*	**-1,34552**	0,396369	7 - 8		0,349512	0,396369
1 - 13		-0,15686	0,396369	7 - 9		-0,0174985	0,396369
1 - 14	*	**-0,553586**	0,396369	7 - 10	*	**-0,453389**	0,396369
1 - 15	*	**-1,03275**	0,396369	7 - 11	*	**-1,03869**	0,396369
1 - 16		-0,276453	0,396369	7 - 12	*	**-1,48283**	0,396369
1 - 17	*	**-0,475749**	0,396369	7 - 13		-0,294171	0,396369
1 - 18	*	**-0,971367**	0,396369	7 - 14	*	**-0,690897**	0,396369
2 - 3		-0,145027	0,396369	7 - 15	*	**-1,17006**	0,396369
2 - 4		0,251777	0,396369	7 - 16	*	**-0,413764**	0,396369

2 - 5	*	**0,528388**	0,396369	7 - 17	*	**-0,613061**	0,396369
2 - 6		0,0799534	0,396369	7 - 18	*	**-1,10868**	0,396369
2 - 7		0,194986	0,396369	8 - 9		-0,36701	0,396369
2 - 8	*	**0,544497**	0,396369	8 - 10	*	**-0,802901**	0,396369
2 - 9		0,177487	0,396369	8 - 11	*	**-1,3882**	0,396369
2 - 10		-0,258403	0,396369	8 - 12	*	**-1,83234**	0,396369
2 - 11	*	**-0,843702**	0,396369	8 - 13	*	**-0,643682**	0,396369
2 - 12	*	**-1,28784**	0,396369	8 - 14	*	**-1,04041**	0,396369
2 - 13		-0,0991852	0,396369	8 - 15	*	**-1,51957**	0,396369
2 - 14	*	**-0,495911**	0,396369	8 - 16	*	**-0,763276**	0,396369
2 - 15	*	**-0,975073**	0,396369	8 - 17	*	**-0,962572**	0,396369
2 - 16		-0,218779	0,396369	8 - 18	*	**-1,45819**	0,396369
2 - 17	*	**-0,418075**	0,396369	9 - 10	*	**-0,43589**	0,396369
2 - 18	*	**-0,913692**	0,396369	9 - 11	*	**-1,02119**	0,396369
3 - 4	*	**0,396804**	0,396369	9 - 12	*	**-1,46533**	0,396369
3 - 5	*	**0,673415**	0,396369	9 - 13		-0,276672	0,396369
3 - 6		0,22498	0,396369	9 - 14	*	**-0,673399**	0,396369
3 - 7		0,340012	0,396369	9 - 15	*	**-1,15256**	0,396369
3 - 8	*	**0,689524**	0,396369	9 - 16		-0,396266	0,396369
3 - 9		0,322514	0,396369	9 - 17	*	**-0,595562**	0,396369
3 - 10		-0,113376	0,396369	9 - 18	*	**-1,09118**	0,396369
3 - 11	*	**-0,698675**	0,396369	10 - 11	*	**-0,585299**	0,396369
3 - 12	*	**-1,14282**	0,396369	10 - 12	*	**-1,02944**	0,396369
3 - 13		0,0458417	0,396369	10 - 13		0,159218	0,396369
3 - 14		-0,350885	0,396369	10 - 14		-0,237508	0,396369
3 - 15	*	**-0,830046**	0,396369	10 - 15	*	**-0,71667**	0,396369
3 - 16		-0,0737517	0,396369	10 - 16		0,0396248	0,396369
3 - 17		-0,273048	0,396369	10 - 17		-0,159672	0,396369
3 - 18	*	**-0,768665**	0,396369	10 - 18	*	**-0,655289**	0,396369
4 - 5		0,276611	0,396369	11 - 12	*	**-0,444141**	0,396369
4 - 6		-0,171823	0,396369	11 - 13	*	**0,744517**	0,396369
4 - 7		-0,0567911	0,396369	11 - 14		0,347791	0,396369
4 - 8		0,292721	0,396369	11 - 15		-0,131371	0,396369
4 - 9		-0,0742896	0,396369	11 – 16	*	**0,624924**	0,396369
4 - 10	*	**-0,51018**	0,396369	11 – 17	*	**0,425627**	0,396369
4 - 11	*	**-1,09548**	0,396369	11 – 18		-0,06999	0,396369
4 - 12	*	**-1,53962**	0,396369	12 – 13	*	**1,18866**	0,396369
4 - 13		-0,350962	0,396369	12 – 14	*	**0,791932**	0,396369
4 - 14	*	**-0,747688**	0,396369	12 – 15		0,312771	0,396369
4 - 15	*	**-1,22685**	0,396369	12 – 16	*	**1,06907**	0,396369
4 - 16	*	**-0,470555**	0,396369	12 – 17	*	**0,869769**	0,396369
4 - 17	*	**-0,669852**	0,396369	12 – 18		0,374151	0,396369
4 - 18	*	**-1,16547**	0,396369	13 – 14	*	**-0,396726**	0,396369
5 - 6	*	**-0,448435**	0,396369	13 – 15	*	**-0,875888**	0,396369
5 - 7		-0,333402	0,396369	13 – 16		-0,119593	0,396369
5 - 8		0,0161093	0,396369	13 – 17		-0,31889	0,396369
5 - 9		-0,350901	0,396369	13 – 18	*	**-0,814507**	0,396369
5 - 10	*	**-0,786791**	0,396369	14 – 15	*	**-0,479162**	0,396369
5 - 11	*	**-1,37209**	0,396369	14 – 16		0,277133	0,396369
5 – 12	*	**-1,81623**	0,396369	14 – 17		0,0778365	0,396369
5 – 13	*	**-0,627573**	0,396369	14 – 18	*	**-0,417781**	0,396369
5 – 14	*	**-1,0243**	0,396369	15 – 16	*	**0,756295**	0,396369
5 – 15	*	**-1,50346**	0,396369	15 – 17	*	**0,556998**	0,396369
5 – 16	*	**-0,747167**	0,396369	15 – 18		0,0613808	0,396369
5 – 17	*	**-0,946463**	0,396369	16 – 17		-0,199296	0,396369
5 – 18	*	**-1,44208**	0,396369	16 – 18	*	**-0,694914**	0,396369
6 – 7		0,115032	0,396369	17 – 18	*	**-0,495617**	0,396369
6 – 8	*	**0,464544**	0,396369				

Este resultado é validado pelo gráfico de médias e intervalos, mostrado na Figura 8.

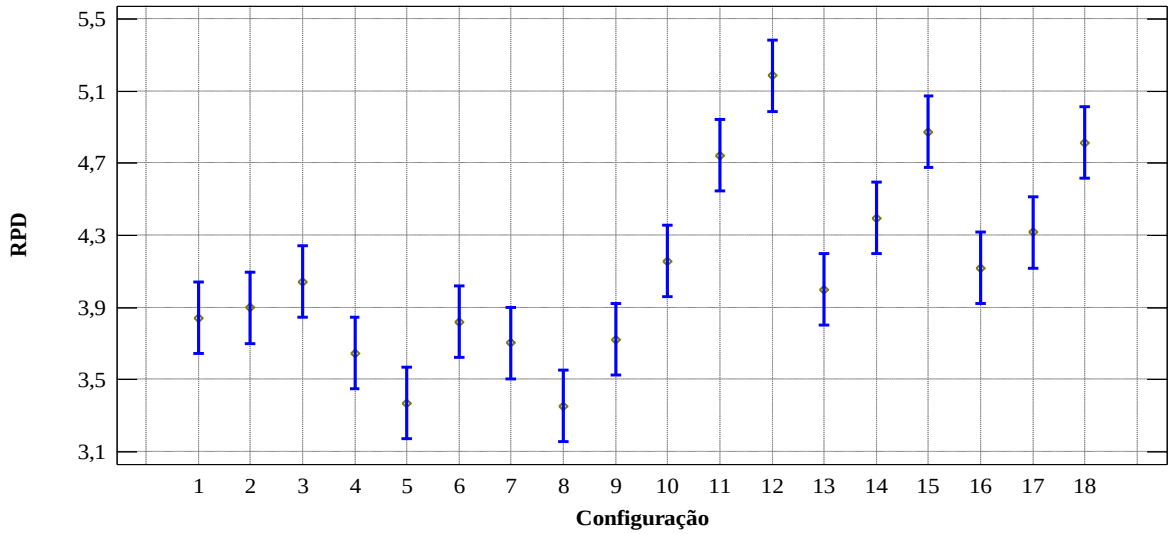

Figura 8: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para as configurações dos parâmetros do algoritmo ILS

A Figura 8 mostra as médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD, com nível de confiança de 95%. Quando um intervalo de um objeto analisado não é sobreposto aos outros intervalos, conclui-se que há diferença significativa entre as médias. Pode-se observar que as configurações 4, 5, 7, 8 e 9 se interceptam, sugerindo que não há diferenças significativas entre elas, como apresentado nos testes anteriores. As configurações 5 e 8 apresentam resultados muito próximos; porém, a configuração 8 possui a menor média para o RPD, conforme mostrado na Tabela 11. Assim, conclui-se que a configuração número 8, constituída pelos valores 2, 25 e 0.3, respectivamente, para os parâmetros T, d_{max} e β, é que determina um melhor resultado para o algoritmo ILS e é esta configuração que será usada para o experimento de comparação dos algoritmos heurísticos.

2.4.4.5 Calibração de parâmetros do IG

Para o algoritmo IG foram analisados dois parâmetros:

- k : correspondente ao tamanho do bloco que será removido da sequência;
- β : corresponde a probabilidade de aceitação de uma solução pior.

Os seguintes conjuntos de valores foram testados: $k \in \{2, 4, 6\}$ e $\beta \in \{0, 0.3, 0.6\}$. Dessa forma, foram testadas 9 configurações de parâmetros, conforme mostra a Tabela 14.

Tabela 14: Combinação dos parâmetros k e β para o algoritmo IG

Configuração	k	β
1	2	0
2	2	0.3
3	2	0.6
4	4	0
5	4	0.3
6	4	0.6
7	6	0
8	6	0.3
9	6	0.6

O teste de variância aplicado sobre os desvios percentuais relativos das rodadas mostra que os dados não possuem homocedasticidade suficiente para o uso ANOVA, como pode ser visto através do teste de *Levene's*, apresentado na Tabela 15. O valor da coluna "*P-Value*" ficou inferior a 0.05, indicando que há uma diferença significativa entre os desvios padrões, o que inviabiliza o uso da ANOVA.

Tabela 15: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de calibração dos parâmetros do IG

	Test	*P-Value*
Levene's	5,15386	**0,00000206954**

Por este motivo utilizou-se o teste não paramétrico de *Kruskal-Wallis*. O resultado da aplicação do mesmo foi um valor "*P-value*" igual a 0 (zero), que por ser inferior a 0.05, indica que há diferenças estatísticas significativas entre as configurações testadas.

Para visualizar quais são essas configurações estatisticamente diferentes, é aplicado o teste de múltiplas comparações (apresentado na Tabela 16). Neste teste cada par de configuração é comparado com nível de confiança de 95% e a coluna "Diferença" mostra a média das amostras da primeira versão menos as da segunda. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer par de versão no qual o valor absoluto da diferença exceda o limite, indica que as versões são, de forma significativa, estatisticamente diferentes, com nível de confiança de 95%. Na tabela 16, a existência dessa diferença é indicado por um (*) na coluna "Significante". É possível perceber que a configuração 5 possui diferença significante em relação a todas as outras configurações.

Tabela 16: Resultado do teste de múltiplas comparações entre as diferentes configurações do algoritmo ILS

Comparação	***Significante***	***Diferença***	***+/- Limite***	***Comparação***	***Significante***	***Diferença***	***+/- Limite***
1 - 2	*	**2,23607**	0,364183	3 - 7		-0,339341	0,364183
1 - 3	*	**2,31743**	0,364183	3 - 8		-0,0755089	0,364183
1 - 4	*	**1,45672**	0,364183	3 - 9	*	**-0,643019**	0,364183
1 - 5	*	**2,72567**	0,364183	4 - 5	*	**1,26895**	0,364183
1 - 6	*	**2,30388**	0,364183	4 - 6	*	**0,847159**	0,364183
1 - 7	*	**1,97809**	0,364183	4 - 7	*	**0,521374**	0,364183
1 - 8	*	**2,24193**	0,364183	4 - 8	*	**0,785206**	0,364183
1 - 9	*	**1,67442**	0,364183	4 - 9		0,217696	0,364183
2 - 3		0,0813671	0,364183	5 - 6	*	**-0,421794**	0,364183
2 - 4	*	**-0,779348**	0,364183	5 - 7	*	**-0,747579**	0,364183
2 - 5	*	**0,489605**	0,364183	5 - 8	*	**-0,483747**	0,364183
2 - 6		0,0678113	0,364183	5 - 9	*	**-1,05126**	0,364183
2 - 7		-0,257974	0,364183	6 - 7		-0,325785	0,364183
2 - 8		0,00585817	0,364183	6 - 8		-0,0619531	0,364183
2 - 9	*	**-0,561652**	0,364183	6 - 9	*	**-0,629463**	0,364183
3 - 4	*	**-0,860715**	0,364183	7 - 8		0,263832	0,364183
3 - 5	*	**0,408238**	0,364183	7 - 9		-0,303678	0,364183
3 - 6		-0,0135558	0,364183	8 - 9	*	**-0,56751**	0,364183

O resultado do teste de múltiplas comparações apresenta as diferenças entre as configurações, mas não mostra o resultado médio obtido por cada configuração, por isso, é apresentado na Tabela 17, o RPD médio alcançado por cada configuração, com base no mesmo teste de múltiplas comparações. Também é possível visualizar os grupos de configurações que possuem médias homogêneas, ou seja, aquelas que não possuem diferenças estatísticas

significantes entre si. Isso pode ser visto através da coluna “Grupos homogêneos”: as configurações que possuem o “X” na mesma direção vertical pertencem ao mesmo grupo.

Tabela 17: Apresentação do resultado médio da calibração dos parâmetros do algoritmo IG

Configuração	*Amostra*	*RPD médio*	*Grupos Homogêneos*
5	375	2,77881	X
3	375	3,18705	X
6	375	3,20060	X
8	375	3,26256	X
2	375	3,26841	X
7	375	3,52639	XX
9	375	3,83007	XX
4	375	4,04776	X
1	375	5,50448	X

O mesmo resultado pode ser visto através do gráfico de médias e intervalos (Figura 9). Nele são mostrados as médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95%. Quando o intervalo de um objeto analisado não sobrepõe aos outros intervalos, conclui-se que há diferenças significativas entre as médias. Pode-se observar, então que a configuração 5 é a que possiu menor intervalo (média) e não é interceptada por nenhuma outra faixa.

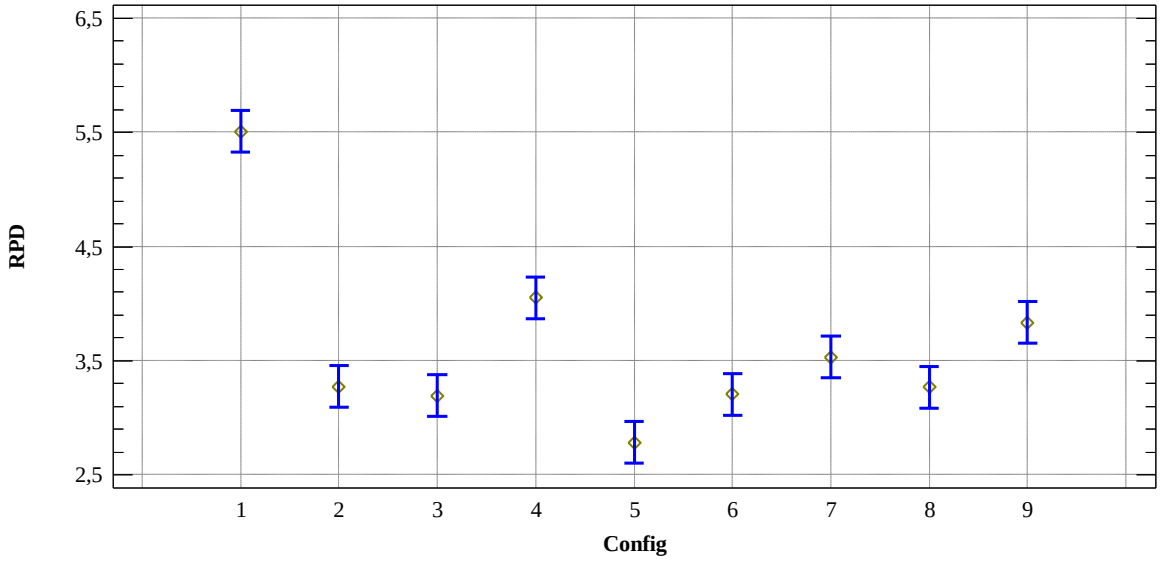

Figura 9: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para as configurações dos parâmetros do algoritmo IG

Assim, pode-se concluir que a configuração número 5, constituída pelos valores 4 e 0.3, respectivamente para os parâmetros k e β, é que determina um melhor resultado para o algoritmo IG.

2.4.4.6 Calibração de parâmetros do ILS-IG

Para calibração do algoritmo ILS-IG, os seguintes parâmetros foram analisados:

- T: Tamanho inicial da perturbação;
- d_{max}: Número de iterações sem melhora realizada pelo algoritmo antes do nível da pertubação aumentar;

- β : corresponde à probabilidade de aceitação de uma solução pior;
- k: tamanho do bloco de tarefas que será desconectado no procedimento DC*.

Os seguintes conjuntos de valores foram adotados: $T \in \{2, 4\}$, $d_{max} \in \{15, 20, 25\}$ e $\beta \in \{0, 0.3, 0.6\}$ e $k \in \{2, 4, 6\}$. As combinações dos valores dos quatro parâmetros geram 54 diferentes configurações para o algoritmo. A identificação destas configurações são mostradas na Tabela 18 a seguir.

Após a execução do algoritmo com cada uma das configurações, o teste de variância foi aplicado com o intuíto de verificar a pressuposição de homocedasticidade para o uso da ANOVA. Como pode ser visto através do teste de *Levene's*, apresentado na Tabela 19, o valor da coluna "*P-Value*" ficou inferior a 0.05, indicando que há uma diferença significativa entre os desvios padrões, o que inviabiliza o uso da ANOVA.

Utiliza-se então o teste não paramétrico de *Kruskal-Wallis*. O valor "*P-value*" resultante do teste é 0.16075. Como trata-se de um valor inferior a 0.05, entende-se que há diferenças significativas entre as configurações testadas.

Tabela 18: Combinação dos parâmetros T, d_{max}, β e k para o algoritmo ILS-IG

Configuração	T	d_{max}	β	k	**Configuração**	T	d_{max}	β	k
1	2	15	0	2	28	4	15	0	4
2	2	15	0,3	2	29	4	15	0,3	4
3	2	15	0,6	2	30	4	15	0,6	4
4	2	20	0	2	31	4	20	0	4
5	2	20	0,3	2	32	4	20	0,3	4
6	2	20	0,6	2	33	4	20	0,6	4
7	2	25	0	2	34	4	25	0	4
8	2	25	0,3	2	35	4	25	0,3	4
9	2	25	0,6	2	36	4	25	0,6	4
10	4	15	0	2	37	2	15	0	6
11	4	15	0,3	2	38	2	15	0,3	6
12	4	15	0,6	2	39	2	15	0,6	6
13	4	20	0	2	40	2	20	0	6
14	4	20	0,3	2	41	2	20	0,3	6
15	4	20	0,6	2	42	2	20	0,6	6
16	4	25	0	2	43	2	25	0	6
17	4	25	0,3	2	44	2	25	0,3	6
18	4	25	0,6	2	45	2	25	0,6	6
19	2	15	0	4	46	4	15	0	6
20	2	15	0,3	4	47	4	15	0,3	6
21	2	15	0,6	4	48	4	15	0,6	6
22	2	20	0	4	49	4	20	0	6
23	2	20	0,3	4	50	4	20	0,3	6
24	2	20	0,6	4	51	4	20	0,6	6
25	2	25	0	4	52	4	25	0	6
26	2	25	0,3	4	53	4	25	0,3	6
27	2	25	0,6	4	54	4	25	0,6	6

Tabela 19: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de calibração dos parâmetros do IG

	Test	*P-Value*
Levene's	0,717306	**1,0**

Para verificar quais são os objetos que apresentam diferenças, foi aplicado o teste de múltiplas comparações, que compara cada par de médias, com nível de confiança de 95%, e mostra se o resultado corresponde a uma diferença significativa entre os objetos. A saída padrão do teste, que mostra a comparação de cada par média, foi omitida, devido o tamanho da tabela resultante (1000 linhas), mas uma saída alternativa, baseada no agrupamento dos objetos que possuem médias homogêneas é apresentada na Tabela 20.

Tabela 20: Apresentação do resultado médio da calibração dos parâmetros do algoritmo IG

Configuração	*Amostra*	*RPD médio*	*Grupos Homogêneos*
52	375	3,27198	X
43	375	3,28342	X
40	375	3,36924	XX
34	375	3,37543	XXX
26	375	3,38353	XXX
25	375	3,39889	XXX
22	375	3,41806	XXXX
37	375	3,42485	XXXX
49	375	3,4641	XXXXX
31	375	3,49792	XXXXXX
19	375	3,50897	XXXXXX
23	375	3,51968	XXXXXX
46	375	3,61112	XXXXXXX
20	375	3,65315	XXXXXXXX
24	375	3,70158	XXXXXXXX
28	375	3,70247	XXXXXXXX
35	375	3,73569	XXXXXXXXX
38	375	3,77483	XXXXXXXXX
27	375	3,77681	XXXXXXXXX
29	375	3,81428	XXXXXXXXX
21	375	3,83527	XXXXXXXXX
32	375	3,85268	XXXXXXXXXX
53	375	3,88832	XXXXXXXXXX
44	375	3,97679	XXXXXXXXXX
33	375	3,98667	XXXXXXXXXX
41	375	3,99976	XXXXXXXXXX
6	375	4,00884	XXXXXXXXXX
50	375	4,05733	XXXXXXXXXX
36	375	4,07692	XXXXXXXXX
30	375	4,10996	XXXXXXXXX
47	375	4,13079	XXXXXXXXXX
9	375	4,14641	XXXXXXXXX
45	375	4,15331	XXXXXXXXX
8	375	4,17603	XXXXXXXXXX
7	375	4,20464	XXXXXXXXX
4	375	4,22403	XXXXXXXX
5	375	4,22615	XXXXXXXX
2	375	4,24542	XXXXXXXX
18	375	4,29162	XXXXXXXX
3	375	4,30003	XXXXXXXX
42	375	4,33606	XXXXXXXX
39	375	4,35959	XXXXXXXX
17	375	4,4547	XXXXXXX
14	375	4,50206	XXXXXX
15	375	4,51487	XXXXXX
48	375	4,51664	XXXXX
16	375	4,53495	XXXXX
54	375	4,57927	XXXX

13	375	4,63437	XXX
10	375	4,64701	XXX
51	375	4,66819	XX
1	375	4,66926	XX
11	375	4,70141	X
12	375	4,70324	X

A Tabela 20 mostra o resultado do RPD médio alcançado por cada configuração, através da coluna “RPD médio”. Também, é possível visualizar os grupos de configurações que possuem médias homogêneas, ou seja, aquelas que não possuem diferenças estatísitcas significantes entre si. Isso pode ser visto através da coluna “Grupos homogêneos”: as configurações que possuem o “X” na mesma direção vertical pertencem ao mesmo grupo. É obeservado que as melhores configurações são: 52, 43, 40, 34, 26, 25, 22, 37, 49, 31, 19, 23, 46 e 20. Apesar de não haver diferenças significativas entre elas, a configuração 52 é a que obteve os melhores resultados.

Para faciltar a visualização a respeito das melhores configurações e das diferenças estatísticas significativas, foi gerado o gráfico de médias e intervalos, apresentado na Figura 10, que mostra as médias resultantes do teste Tukey da Diferença Honestamente Significativa (HSD) com nível de confiança de 95%.

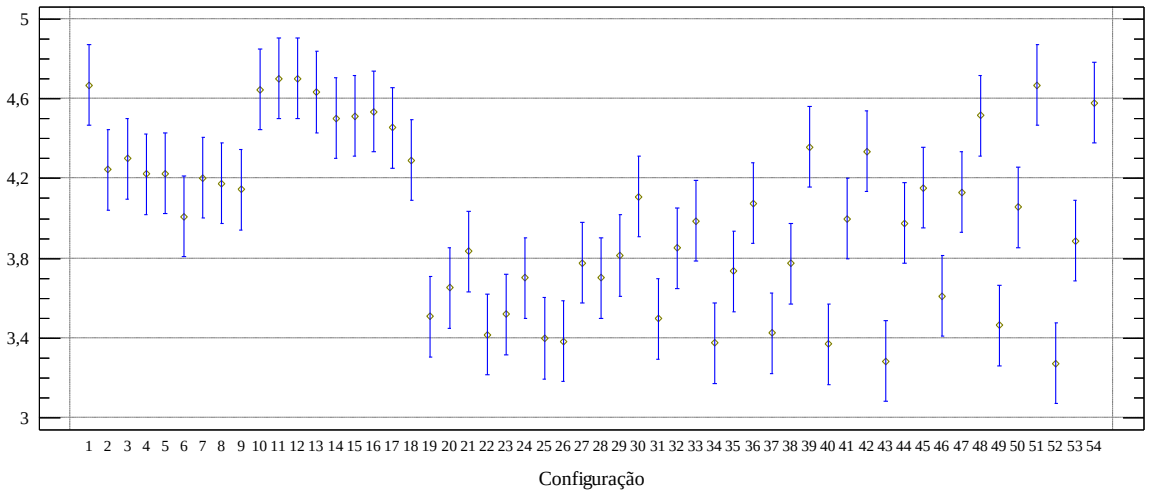

Figura 10: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para as configurações dos parâmetros do algoritmo ILS-IG

Assim como mostrado na Tabela 20, é possível observar que as configurações 19, 20, 22, 23, 25, 26, 31, 34, 37, 40, 43, 46, 49 e 52 têm seus intervalos sobrepostos uns aos outros indicando que não há diferenças significantes entre elas. Porém, deseja-se selecionar a configuração que obteve o menor RPD médio. Para isso, o gráfico novamente foi gerado, porém

apresentando somente as 14 configurações referidas. O gráfico de médias e intervalos é apresentado na Figura 11.

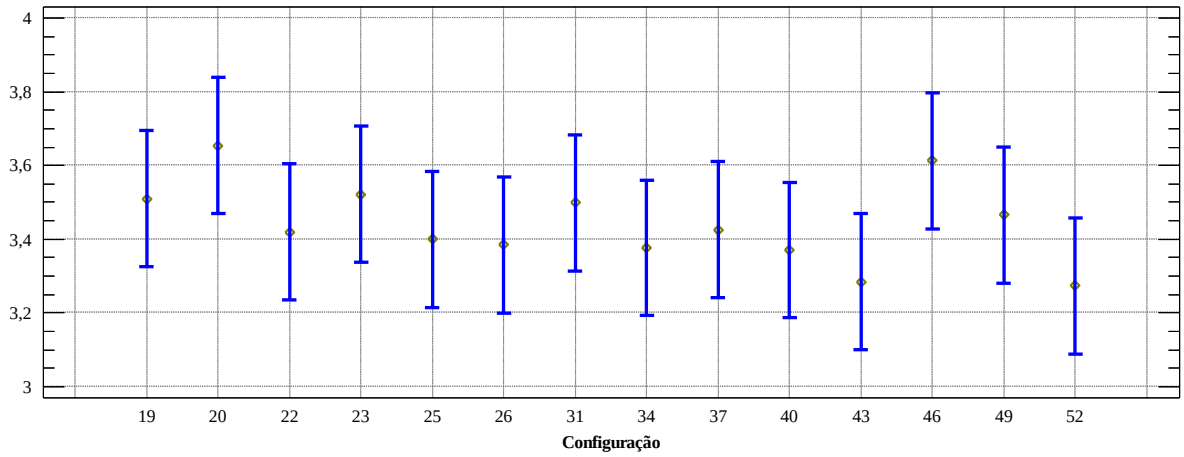

Figura 11: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para as melhores configurações de parâmetros do algoritmo ILS-IG: 19, 20, 22, 23, 25, 26, 31, 34, 37, 40, 43, 46, 49 e 52

Observa-se que as configurações 43 e 52 são as que possuem as menores médias e que têm resultados parecidos. A primeira possui média 3.28342 enquanto a segunda de 3.27198, resultando em uma diferença de apenas 0.01144, como pode ser visto na Tabela 19. Apesar da pequena diferença, a configuração 52, que corresponde a combinação de parâmetros T=4, d_{max}=25, β=0 e k=6, foi a escolhida por obter o melhor resultado.

2.4.5 Experimento computacional para comparação das soluções heurísticas

Nesta seção, comparamos as soluções heurísticas propostas neste trabalho, juntamente com um algoritmo da literatura, PSA, proposto por Allahverdi e Aydilek (2013). Este experimento foi realizado em duas etapas, descritas a seguir:

- **Etapa 1:** Avaliação de desempenho em grandes instâncias;
- **Etapa 2:** Avaliação de desempenho em pequenas instâncias;

Para a etapa 1 deste experimento, todos os algoritmos foram rodados sobre todo o conjunto de grandes instâncias (150). Cada algoritmo foi rodado 10 vezes para cada instância, totalizando assim, 1500 execuções. O critério de parada para todos os algoritmos foi o tempo de execução, deteminado pela fórmula: $n \times m \times c$, onde c=100.

A Tabela 21 mostra o resultado do experimento definido na etapa 1. É exibido o RPD médio e o tempo médio de execução (em segundos), alcançado por cada algoritmo, sobre cada

combinação de problemas $n \times m$. Os melhores resultados estão destacados na tabela com formatação em negrito e itálico.

Tabela 21: Resultado médio da execução dos algoritmos sobre as instâncias de grande porte agrupadas por *n x m*

Instância		**PSA**		**ILS**		**IG**		**ILS-IG**		**GRASP-RVND**	
n	**m**	**Tempo**	**RPD**	**Tempo**	**RPD**	**Tempo**	**RPD**	**Tempo**	**RPD**	**Tempo**	**RPD**
30	5	16	5,86	15	1,07	15	0,31	15	***0,20***	15	1,19
30	10	32	5,37	30	0,73	30	0,29	30	***0,16***	30	1,35
30	20	64	5,41	60	0,73	60	0,22	60	***0,15***	60	0,98
50	5	27	8,32	25	3,24	25	2,20	25	***1,73***	25	5,60
50	10	54	6,19	50	2,96	50	1,76	50	***1,35***	50	4,54
50	20	102	6,62	100	3,20	100	1,86	100	***1,48***	100	4,54
80	5	43	12,73	40	5,03	40	3,42	40	***2,93***	41	8,81
80	10	80	9,47	80	4,06	80	2,74	80	***2,58***	81	7,20
80	20	160	6,89	160	3,06	160	1,85	160	***1,72***	162	5,47
100	5	54	13,29	50	4,94	50	4,15	50	***2,89***	51	10,05
100	10	102	11,80	100	4,64	100	3,60	100	***2,77***	102	8,28
100	20	204	7,42	200	2,61	200	2,34	200	***1,92***	204	5,59
200	5	95	14,95	101	4,57	101	5,50	101	***4,56***	127	8,33
200	10	187	14,88	201	***3,83***	201	5,25	201	5,23	250	7,15
200	20	384	10,92	402	***2,56***	403	3,21	402	3,33	494	4,47
Média		**107**	**9,34**	**108**	**3,15**	**108**	**2,58**	**108**	***2,20***	**119**	**5,57**

A Tabela 22 mostra o RPD médio e o tempo médio de execução (em segundos), obtidos em cada conjunto de instância de tamanho *n*. Os melhores resultados estão destacados em negrito e itálico.

Tabela 22: Resultado médio da execução dos algoritmos sobre as instâncias de grande porte agrupadas por número de tarefas

Inst.	**PSA**		**ILS**		**IG**		**ILS-IG**		**GRASP-RVND**	
n	**Tempo**	**RPD**	**Tempo**	**RPD**	**Tempo**	**RPD**	**Tempo**	**RPD**	**Tempo**	**RPD**
30	37	5,55	35	0,84	35	0,27	35	***0,17***	35	1,17
50	61	7,05	58	3,13	58	1,94	58	***1,52***	58	4,89
80	95	9,69	93	4,05	93	2,67	93	***2,41***	94	7,16
100	120	10,84	117	4,06	117	3,37	117	***2,53***	119	7,97
200	222	13,58	235	***3,65***	235	4,65	235	4,37	290	6,65
Média	**107**	**9,34**	**108**	**3,15**	**108**	**2,58**	**108**	***2,20***	**119**	**5,57**

Pode-se reparar que o ILS-IG foi o algoritmo que obteve os melhores resultados, exceto para as instâncias de problemas com 200 tarefas e com mais de 5 máquinas. Para estas instâncias, o melhor resultado foi obtido pelo ILS que, em média, ficou 16% melhor que o ILS-IG. Mas considerando a média geral, o ILS-IG obtem resultados 43.18% melhores que o ILS, 17.27% melhor que o IG, 153% melhor que o GRASP-RVND e 324% melhor que o PSA.

Realizou-se uma análise estatística sobre os resultados obtidos com o intuito de verificar se há diferenças significantes entre eles. Primeiramente, avaliou-se a pressuposição de

homocedasticidade para o uso da ANOVA, porém o valor "*P-value*" retornado pelo teste de *Levene's* foi 0 (zero), conforme pode ser visto na Tabela 23, indicando que a pressuposição é falha, visto que é um valor inferior a 0.05

Tabela 23: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados obtidos pela execução dos algoritmos heurísticos

	Test	*P-Value*
Levene's	284,146	**0**

Por este motivo, utiliza-se o teste não paramétrico de *Kruskal-Wallis*. O valor "*P-value*" resultante do teste é 0. Por se tratar de um valor inferior a 0.05, entende-se que há diferenças significativas entre as configurações testadas. Para visualizar quais são os objetos estatisticamente diferentes, aplica-se o teste de múltiplas comparações. O resultado do teste é mostrado na Tabela 24. A coluna "Diferença" mostra a média das amostras da primeira versão menos as da segunda. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer par de versão no qual o valor absoluto da diferença exceda o limite, indica que as versões são, de forma significativa, estatisticamente diferentes, com nível de confiança de 95%. Na tabela 24, a existência dessa diferença é indicado por um (*) na coluna "Significante".

Tabela 24: Resultado do teste de múltiplas comparações entre os algoritmos heurísticos propostos para instâncias de grande porte

Comparação	Significante	Diferença	+/- Limite
GRASP-RVND - IG	*	**3,13169**	0,252417
GRASP-RVND - ILS	*	**2,54764**	0,252417
GRASP-RVND - ILS-IG	*	**3,54696**	0,252417
GRASP-RVND - PSA	*	**-4,21121**	0,252417
IG - ILS	*	**-0,584056**	0,252417
IG - ILS-IG	*	**0,415264**	0,252417
IG - PSA	*	**-7,3429**	0,252417
ILS - ILS-IG	*	**0,99932**	0,252417
ILS - PSA	*	**-6,75885**	0,252417
ILS-IG - PSA	*	**-7,75817**	0,252417

É notável que existem diferenças significativas entre todos os pares comparados. Isso mostra que cada algoritmo está em um grupo diferente, em termos de resultados estatísticos. Para facilitar a visualização da diferença entre os resultados gerados, a Figura 12, mostra o gráfico de médias e intervalos. Nele são mostrados as médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD, com nível de confiança de 95%. Quando o intervalo de um objeto analisado não sobrepõe aos outros intervalos, conclui-se que há diferença significativa entre as médias. Pode-se perceber que nenhum dos algoritmos se interceptam, validando assim o resultado mostrado pelo teste de múltiplas comparações.

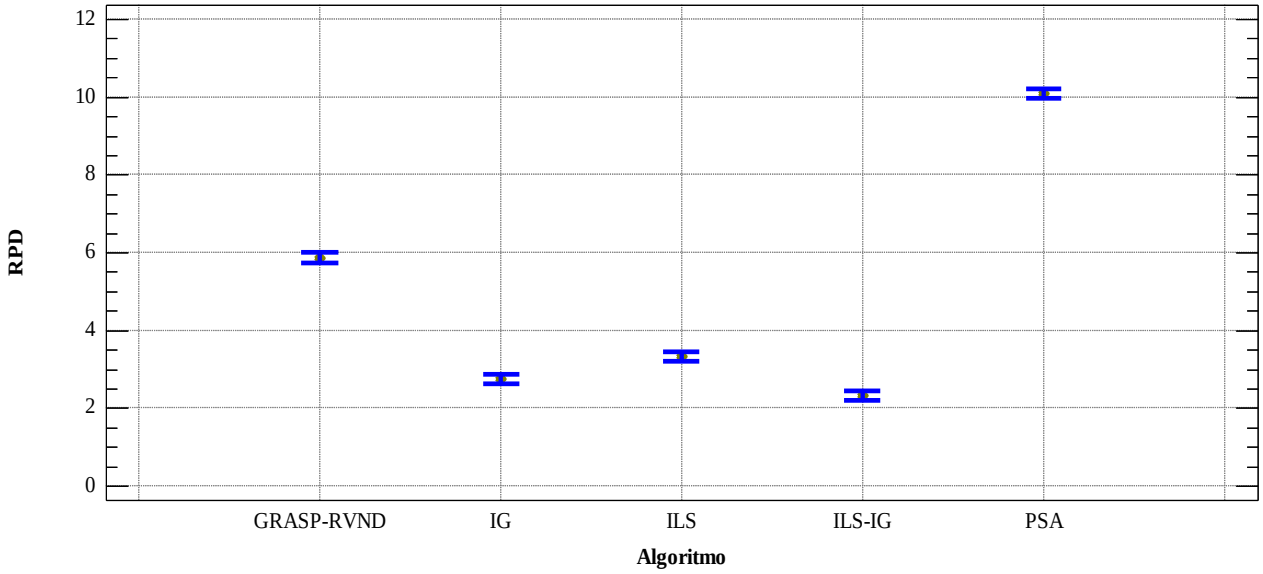

Figura 12: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos resultados obtidos pelos algoritmos GRASP-RVND, IG, ILS, ILS-IG e PSA para instâncias de grande porte

Ao observar a Figura 12 percebe-se que os melhores resultados, em média, são alcançados, respectivamente, pelos algoritmos ILS-IG, IG, ILS, GRASP-RVND e PSA. Porém o algoritmo ILS obteve os melhores resultados para as instâncias de maior tamanho (200 tarefas). O GRASP-RVND e PSA são os que obtiveram os maiores desvios percentuais relativos, apresentando uma diferença "alta" para os demais algoritmos.

A Figura 13 mostra o mesmo gráfico de intervalo de médias, porém suprimindo o GRASP-RVND e o PSA, proporcionando assim uma visão mais clara sobre a diferença dos resultados obtidos pelos três melhores algoritmos (ILS-IG, IG e ILS).

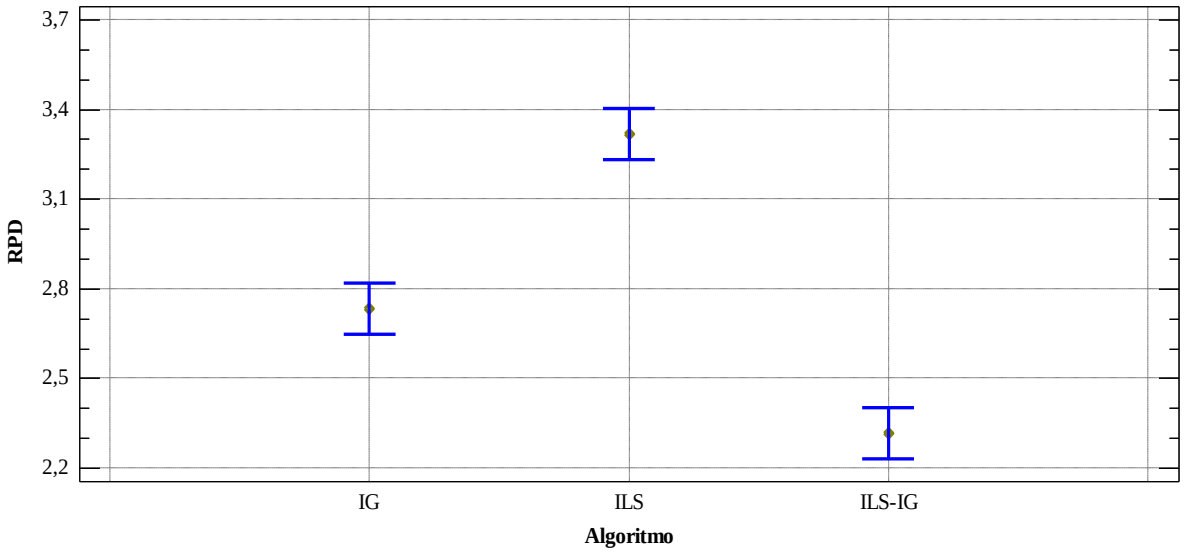

Figura 13: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos resultados obtidos pelos algoritmos IG, ILS e ILS-IG

A etapa 2 deste experimento avalia o desempenho dos algoritmos em pequenas instâncias. Para essa avaliação, todos os algoritmos foram rodados sobre todo o conjunto de pequenas instâncias (120) e os resultados obtidos foram comparados com o resultado ótimo gerado pela execução do modelo matemático no IBM ILOG CPLEX[2] versão 12.4.

Para execução do modelo matemático, o CPLEX foi configurado para retornar o melhor resultado alcançado em até 2 horas. O apêndice A, através da Tabela 39, mostra os resultados obtidos por todos os algoritmos em cada uma das instâncias de pequeno porte, juntamente com o resultado obtido pela resolução do modelo matemático através do CPLEX.

O CPLEX não conseguiu encontrar o resultado ótimo, dentro do limite de tempo imposto para as instâncias 93, 96, 101, 102, 104, 106, 110, 113, 115, 116 e 120, todas de tamanho n=14, conforme mostrado na Tabela 39 (apêndice A). Para essas instâncias, os resultados encontrados pelos algoritmos propostos foram superiores aos obtidos pelo CPLEX, com um tempo computacional inferior a 1% do tempo gasto pelo CPLEX (em média). Analisando apenas essas instâncias, os algoritmos propostos obtiveram resultados médios melhores que o CPLEX, cerca de 4,5%, consumindo um tempo de CPU médio de 6,12 segundos, enquanto o CPLEX gastou 7.201,72 segundos (em média).

Os algoritmos ILS e GRASP-RVND alcançaram a solução ótima em todas as 1200 execuções e para todas as 120 instâncias. Enquanto o ILS-IG encontrou em 1170 vezes a solução ótima, o que corresponde a 97.5% das vezes. O IG encontrou a solução ótima em 79.6% (955 vezes) e o PSA em 38.3% das execuções (459 vezes). Outro importante fator a ser destacado é que os algoritmos rodaram (em média) em 33% do tempo médio do CPLEX. Isso mostra que, em um tempo bem menor, os algoritmos ILS, GRASP-RVND e ILS-IG são capazes de encontrar a solução ótima para todas (ou praticamente todas) as instâncias de teste do problema. Essa diferença aumenta ainda mais quando são comparadas apenas as instâncias de tamanho n=14.

A Tabela 25 mostra a média dos resultados encontrados pelos algoritmos para cada tamanho de instância. É exibido o RPD médio em relação a melhor solução conhecida. Além disso, também é exibido o tempo médio (em segundos) gasto por cada algoritmo.

[2] Pacote de softwares para resolução de modelos matemáticos de programação linear, inteira e mista. Mais detalhes estão disponíveis em: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Avaliando a Tabela 25, percebe-se que ILS-IG não conseguiu alcançar a solução ótima apenas para instâncias com tamanho 8 x 2, porém obteve um RPD médio próximo à zero.

Tabela 25: Resultados médios alcançados pelos algoritmos para as instâncias de pequeno porte

Instancia n x m	PSA		ILS		IG		ILS-IG		GRASP-RVND	
	RPD Médio	Tempo Médio	RPD Médio	Tempo Médio	RPD Médio	Tempo Médio	RPD Médio	Tempo Médio	RPD Médio	Tempo Médio
08 X 02	0,60	1,99	0,00	1,60	4,61	1,61	0,21	1,61	0,00	1,62
08 X 04	0,78	3,21	0,00	3,20	1,48	3,21	0,00	3,21	0,00	3,22
08 X 06	4,17	5,53	0,00	4,81	0,86	4,81	0,00	4,81	0,00	4,82
10 X 02	2,56	1,95	0,00	2,00	0,00	2,00	0,00	2,00	0,00	2,00
10 X 04	3,46	3,83	0,00	4,00	1,69	4,01	0,00	4,00	0,00	4,01
10 X 06	2,82	5,66	0,00	6,01	3,24	6,01	0,00	6,01	0,00	6,01
12 X 02	2,15	2,33	0,00	2,41	0,25	2,41	0,00	2,41	0,00	2,41
12 X 04	4,33	4,58	0,00	4,81	2,58	4,81	0,00	4,81	0,00	4,81
12 X 06	2,24	6,76	0,00	7,20	0,14	7,20	0,00	7,20	0,00	7,21
14 X 02	2,18	2,71	0,00	2,81	0,39	2,81	0,00	2,81	0,00	2,81
14 X 04	3,98	5,31	0,00	5,60	0,27	5,60	0,00	5,60	0,00	5,60
14 X 06	5,36	8,22	0,00	8,41	0,32	8,41	0,00	8,41	0,00	8,41
Média Geral	**2,55**	**4,34**	**0,00**	**4,41**	**1,32**	**4,41**	**0,02**	**4,41**	**0,00**	**4,41**

Para validar os resultados para instâncias pequenas, também foram realizados testes estatísticos. Novamente a ANOVA foi desconsiderada pelo fato dos dados não possuirem homocedasticidade suficiente, conforme pode ser visto na Tabela 26, onde o teste *Levene's* resulta em um "*P-Value*" inferior a 0.05. Então, novamente, o teste não parâmetrico *Kruskal-Wallis* foi adotado, resultando em um "*P-value*" igual a zero, o que indica que existem diferenças estatisticamente significativas entre os objetos.

O teste de múltiplas comparações foi aplicado e sua saída é mostrada na Tabela 27. É perceptível que não há diferenças significativas entre os algoritmos ILS, ILS-IG e GRASP-RVND, porém estas diferenças existem para os outros algoritmos.

Tabela 26: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de comparação dos algoritmos propostos para instâncias de pequeno porte

	Test	*P-Value*
Levene's	22,959	**0**

Tabela 27: Resultado do teste de múltiplas comparações entre os algoritmos heurísticos propostos para instâncias de pequeno porte

Comparação	***Significante***	***Diferença***	***+/- Limite***
GRASP-RVND - IG	*	**-1,31989**	0,591961
GRASP-RVND - ILS		0	0,591961
GRASP-RVND - ILS-IG		-0,017678	0,591961
GRASP-RVND - PSA	*	**-2,5525**	0,591961
IG - ILS	*	**1,31989**	0,591961
IG - ILS-IG	*	**1,30222**	0,591961
IG - PSA	*	**-1,23261**	0,591961
ILS - ILS-IG		-0,017678	0,591961
ILS - PSA	*	**-2,5525**	0,591961
ILS-IG - PSA	*	**-2,53482**	0,591961

A Figura 14 mostra o gráfico de médias e intervalos obtidos na comparação dos algoritmos sobre o conjunto de instâncias pequenas. Nele são apresentadas as médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD, com nível de confiança de 95%. Quando o intervalo de um objeto analisado não sobrepõe aos outros intervalos, conclui-se que há diferença significativa entre as médias. Assim, percebe-se que o teste de múltiplas comparações é validado, uma vez que os intervalos dos algoritmos ILS, GRASP-RVND e ILS-IG se interceptam indicando que não há diferenças estatísticas entre eles, enquanto para os algoritmos IG e PSA essa diferença é clara.

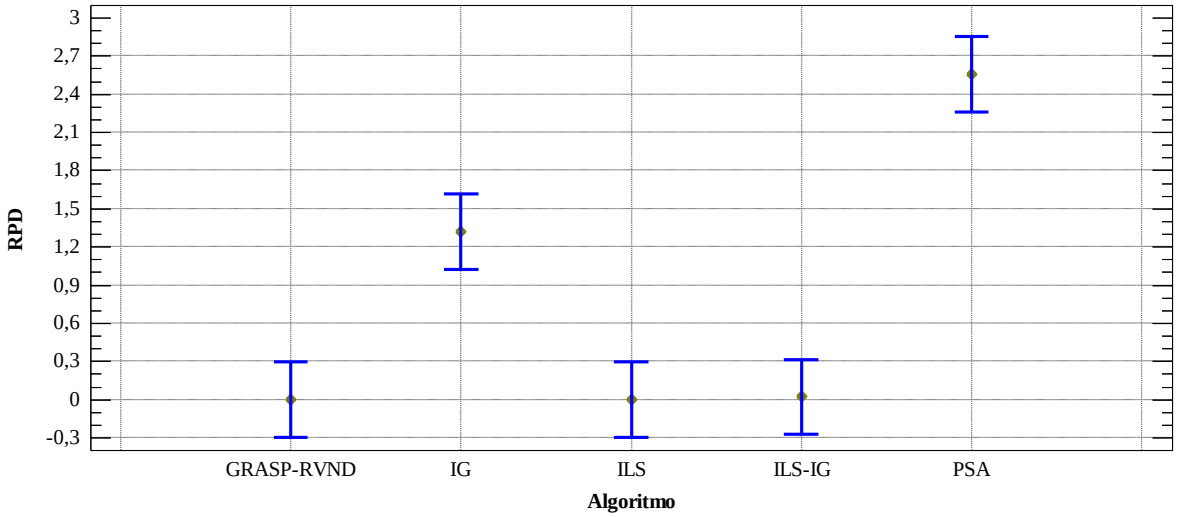

Figura 14: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos resultados obtidos pelos algoritmos GRASP-RVND, IG, ILS, ILS-IG e PSA para instâncias de pequeno porte

2.5 Conclusão da abordagem mono-objetivo

Na primeira parte do trabalho foram examinadas cinco estratégias de resolução para o problema de programação da produção em um ambiente *assembly flowshop* com três estágios, onde o primeiro estágio é responsável pela fabricação de partes da tarefa, constituído por diversas máquinas paralelas não idênticas, o segundo estágio tem a responsabilidade de transportar as partes produzidas e é composto por uma máquina simples, e, por fim, o terceiro estágio é destinado a montagem final da tarefa, também constituído por uma máquina simples. Como objetivo, foi almejada a minimização do atraso total das tarefas. Para deixar o problema ainda mais realístico foram considerados tempos de preparação das máquinas dependentes da sequência de execução das tarefas em todos os estágios do problema.

A literatura mostra que este é um problema complexo de otimização combinatória, com ampla aplicação nas indústrias de manufatura. Em virtude de sua natureza NP-Difícil, este

trabalho focou em estratégias de resolução heurísticas. Para isso, foram produzidos quatro diferentes algoritmos baseados em metaheurísticas: GRASP-RVND, ILS, IG e ILS-IG. Todos os algoritmos foram submetidos a testes de desempenho para definição de aspectos, como construção da solução inicial e busca local, e também para calibração de parâmetros. Os experimentos foram determinados sob a metodologia Desenho de Experimentos e avaliados por diversos testes estatísticos.

Para validar o desempenho dos algoritmos propostos, foi realizado um experimento computacional para comparação com um algoritmo da literatura (PSA), em instâncias de grande e pequeno porte. Nas instâncias de pequeno porte, os resultados são comparados com o resultado ótimo obtido através da resolução de um modelo matemático de programação linear inteira mista (MILP). Todos os experimentos foram realizados sobre condições iguais de tempo de CPU e em um mesmo ambiente computacional.

Em relação as instâncias de grande porte, todos os algoritmos obtiveram desempenho superior ao PSA, sendo o ILS-IG o algoritmo que, em média, obteve os melhores resultados. Porém, para as instâncias com 200 tarefas e com mais de 5 máquinas, o melhores resultados foram encontrados pelo algoritmo ILS. Já quando analisada a execução sobre as instâncias de pequeno porte, os algoritmos ILS e GRASP-RVND conseguiram encontrar a solução ótima para todas as instâncias em todas as execuções. O algoritmo ILS-IG encontrou a solução ótima em 97.5% das vezes. Eles gastaram, em média, cerca 33% do tempo necessário para resolução do modelo matemático.

Portanto, conclui-se que o ILS-IG é o algoritmo que, em média, obtém excelentes resultados, tanto para instâncias de grande porte como para as de pequeno porte, sendo indicado como uma solução padrão para o problema. Já os algoritmos ILS e GRASP-RVND são indicados para instâncias de pequeno porte. Em particular, o ILS também é indicado para instâncias extremamente grandes, com n=200 tarefas e com m>5 máquinas.

2.6 Trabalhos futuros

Como trabalhos futuros, são sugeridos:

a) Aplicação dos algoritmos em instâncias maiores que 200 tarefas;

b) Teste de outras formas de pertubação e outras estratégias de busca local;

c) Explorar outras variações do problema, como, por exemplo, o acréscimo de mais máquinas nos estágios de transporte e montagem ou o uso de um *flowshop* híbrido no primeiro estágio.

d) Melhoria no modelo matemático para resolução de problemas de tamanhos maiores.

2.7 Referências Bibliográficas

Ageel, M. I. (2000). The analysis of variance: Fixed, random, and mixed models. Springer.

Al-Anzi, F. S. e Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. European Journal of Operational Research, vol. 182 no.1, 80-94.

Allahverdi, A., & Al-Anzi, F. S. (2006). Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times. International Journal of Production Research,44(22), 4713-4735.

Allahverdi, A., & Al-Anzi, F. S. (2008). The two-stage assembly flowshop scheduling problem with bicriteria of makespan and mean completion time.The International Journal of Advanced Manufacturing Technology, 37(1-2), 166-177.

Allahverdi, A., & Al-Anzi, F. S. (2009). The two-stage assembly scheduling problem to minimize total completion time with setup times. Computers & Operations Research, 36(10), 2740-2747.

Allahverdi, A., & Aydilek, H. (2014). The two stage assembly flowshop scheduling problem to minimize total tardiness. Journal of Intelligent Manufacturing, 1-13.

Allahverdi, A., Ng, C. T., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 187(3), 985-1032.

Andrés, C. e Hatami, S. (2011). The three stage assembly permutation flowshop scheduling problem. 5th International Conference on Industrial Engineering and Industrial Management (Cartagena, Colombia), 867-875.

Campos, S. C., Arroyo, J. E. C. e Gonçalves, L. B. (2013). Uma heurística GRASP-VND para o problema de sequenciamento de tarefas num ambiente assembly flowshop com três

estágios e tempos de setup dependentes da sequência. Anais do XLV Simpósio Brasileiro de Pesquisa Operacional (Natal-RN, Brasil).

Campos, S. C., Arroyo, J. E. C. (2014). NSGA-II with Iterated Greedy for a bi-objective three-stage assembly flowshop scheduling problem. In Proceeding of the nineteenth annual conference companion on Genetic and evolutionary computation conference companion (GECCO). ACM, 429-436.

Fattahi, P., Hosseini, S. M. H. e Jolai, F. (2013). A mathematical model and extension algorithm for assembly flexible flow shop scheduling problem. The International Journal of Advanced Manufacturing Technology, vol. 65 no.5-8, 787-802.

Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. Operations research letters,8(2), 67-71.

Hariri, A. M. A., & Potts, C. N. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. European Journal of Operational Research, 103(3), 547-556.

Hatami, S., Ebrahimnejad, S., Tavakkoli-Moghaddam, R. e Maboudian, Y. (2010). Two meta-heuristics for three-stage assembly flowshop scheduling with sequence-dependent setup times. The International Journal of Advanced Manufacturing Technology, vol.50 no.9-12, 1153–1164.

Jacob, V. V., Arroyo, J. E. C., dos Santos, A. G. (2013) Heurística busca local iterada para o sequenciamento de tarefas em uma maquina com tempos de preparação dependentes da família. Anais do XLV Simpósio Brasileiro de Pesquisa Operacional (Natal-RN, Brasil).

Koulamas, C., Kyparisis, G. (2001). The three-stage assembly flowshop scheduling problem. Computer & Operation Research, vol. 28, 689–704.

Levitin, A. (2003). Introduction to the design & analysis of algorithms. Addison-Wesley Reading.

Liefooghe, A., Basseur, M., Humeau, J., Jourdan, L. e Talbi, E. G. (2012). On optimizing a bi-objective flowshop scheduling problem in an uncertain environment. Computers & Mathematics with Applications, vol.64 no.12, 3747-3762.

Lopes, H. S., Rodrigues, L. C. A. e Steiner, M. T. A. (2013). Meta-heurísticas em pesquisa operacional, Curitiba, PR: Omnipax.

Lourenço, H. R., Martin, O. e Stutzle T. (2002) Iterated local search. In Handbook of Metaheuristics. Vol. 57 of Operations Research and Management Science. Kluwer Academic Publishers, 2002, 321–353)

Maleki, D. A., Modiri, M., Tavakkoli, M. R. e Seyyedi, I. (2013). A Three-Stage Assembly Flow Shop Scheduling Problem With Blocking And Sequence-Dependent Set Up Times. Journal of Industrial Engineering International, vol. 8 no.1, 1-7.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search.Computers & Operations Research, 24(11), 1097-1100.

Montgomery D. C. (2006) Design and analysis of experiments (7th ed.). New York: Wiley.

Mozdgir, A., Fatemi Ghomi, S. M. T., Jolai, F. e Navaei, J. (2013). Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times. International Journal of Production Research, vol. 51 no.12, 3625-3642.

Naderi, B., Ruiz, R. e Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. Computers & Operations Research, vol. 37 no.2, 236-246.

Navaei, J., Ghomi, S. F., Jolai, F., Shiraqai, M. E., & Hidaji, H. (2013). Two-stage flow-shop scheduling problem with non-identical second stage assembly machines. The International Journal of Advanced Manufacturing Technology, 69(9-12), 2215-2226.

Nawaz, M., Enscore, E. e Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, vol. 11 no.1, 91–95.

Pan, Q. K., Tasgetiren, M. F. e Liang, Y. C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering, vol.55 no.4, 795-816.

Pan, Q. K., & Ruiz, R. (2012). Local search methods for the flowshop scheduling problem with flowtime minimization. European Journal of Operational Research, 222(1), 31-43.

Pan, Q. K., & Wang, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. Omega, 40(2), 218-229.

Pan, Q. K. e Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. Omega, vol. 44, 41-50.

Potts, C. N., Sevast'janov, V. A., Strusevich, V. A., Van Wassenhove, L. N. e Zwaneveld, C. M. (1995) The Two-Stage Assembly Scheduling Problem: Complexity and Approximation. Operations Research, vol. 43 no. 2, 346-355.

Rad S. F., Ruiz R. e Boroojerdian, N. (2009) New high performing heuristics for minimizing makespan in permutation flowshops. OMEGA, The International Journal of Management Science, vol. 37 no 2, 331–345.

Rego, M. F., Souza, M. J. F., Arroyo, J. E. C. (2012). Algoritmos multi-objetivos para o problema de sequenciamento de famílias de tarefas em uma máquina. La Conferencia Latinoamericana en Informática (CLEI), Medellín-Colômbia.

Ribas, I., Leisten, R. e Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research, vol. 37 no. 8, 1439-1454.

Ruiz, R. e Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, vol. 177 no.3, 2033-2049.

Ruiz, R. e Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research, vol.187 no.3, 1143-1159.

Ruiz, R. (2010). Lot-streaming for sequence dependent setup time flowshop problems. In XIV Congreso de Ingeniería de Organización: Donostia-San Sebastián, 8-10 de Septiembre de 2010 (pp. 1739-1747).

Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. European Journal of Operational Research, 205(1), 1-18.

Seidgar, H., Kiani, M., Abedi, M. e Fazlollahtabar, H. (2013). An efficient imperialist competitive algorithm for scheduling in the two-stage assembly flow shop problem. International Journal of Production Research, (ahead-of-print), 1-17.

Shen, W., Wang, L., e Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. Systems, Man, and Cybernetics, Part C: Applications and Reviews. IEEE Transactions on, vol. 36 no.4, 563–577.

Shokrollahpour, E., Zandieh, M., & Dorri, B. (2011). A novel imperialist competitive algorithm for bi-criteria scheduling of the assembly flowshop problem. International Journal of Production Research, 49(11), 3087-3103.

Tajbakhsh, Z., Fattahi, P. e Behnamian, J. (2013). Multi-objective assembly permutation flow shop scheduling problem: a mathematical model and a meta-heuristic algorithm. Journal of the Operational Research Society.

Talbi, E-G. (2009). Metaheuristics: from design to implementation. Jonh Wiley and Sons Inc.

Torabzadeh, E., & Zandieh, M. (2010). Cloud theory-based simulated annealing approach for scheduling in the two-stage assembly flowshop.Advances in Engineering Software, 41(10), 1238-1243.

Tozkapan, A., Kırca, Ö., & Chung, C. S. (2003). A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. Computers & Operations Research, 30(2), 309-320.

Yan, H. S., Wan, X. Q. e Xiong, F. L. (2014). A hybrid electromagnetism-like algorithm for two-stage assembly flow shop scheduling problem. International Journal of Production Research, (ahead-of-print), 1-14.

3 NSGA-II COM ITERATED GREEDY E BUSCA LOCAL PARA RESOLUÇÃO MULTI-OBJETIVO DO PROBLEMA DE SEQUENCIAMENTO ASSEMBLY FLOWSHOP COM TRÊS ESTÁGIOS E TEMPOS DE SETUP DEPENDENTES DA SEQUÊNCIA

3.1 Introdução

O sequenciamento de tarefas é um problema de tomada de decisão que ocorre em diversos tipos de sistemas de manufatura. Esse problema lida com a alocação de recursos escassos para determinadas tarefas durante certos períodos de tempo, com o objetivo de otimizar um ou mais critérios. Os problemas de sequenciamento têm sido exaustivamente estudados desde meados da década de 50 (ALLAHVERDI *ET AL.*, 2008), sendo que, atualmente, os problemas de programação da produção são uns dos mais estudados. Isso ocorre devido a dois fatores: o primeiro diz respeito à sua importância prática, com várias aplicações em diversas indústrias e o segundo aspecto é sobre a dificuldade de resolver os problemas da classe NP-Difícil.

O problema de programação abordado neste trabalho é o problema de sequenciamento de tarefas em um ambiente *assembly flowshop* em três estágios (3sAFS – *three stage assembly flowshop Scheduling*). Neste problema, *n* tarefas (ou produtos) são realizadas em três estágios. No primeiro estágio, as diferentes partes de uma tarefa (produto) são fabricadas de forma independente em máquinas *m* paralelas diferentes (cada trabalho tem *m* partes); no segundo estágio, as peças produzidas são recolhidas e transferidas do local da produção para o local de montagem; e no último estágio, são montados os produtos finais. No segundo e terceiro estágios, há apenas uma única máquina simples.

O ambiente é dito *flowshop* pelo fato dos três estágios estarem dispostos em série. Cada tarefa é processada em cada estágio em uma ordem pré-estabelecida, ou seja, uma tarefa *j* não pode iniciar seu processamento no terceiro estágio sem que antes seja completamente terminada no estágio 2. Isso caracteriza um *flowshop* permutacional. Este problema é NP-Difícil a partir de $m = 1$, o que torna o problema em um *flowshop* com três máquinas regulares, que é sabído ser NP-Difícil (GAREY *ET AL*, 1976).

A fim de tornar o problema ainda mais realista, consideramos tempos de preparação das máquinas dependentes da seqüência em todas as máquinas do primeiro, segundo e terceiro estágios. O tempo de preparação é necessário para configurar a máquina (por exemplo, ajuste

de ferramentas, limpeza, posicionamento de acessórios, inspeção de materiais, entre outros). Dessa forma, o tempo de preparação de uma tarefa *j* depende da tarefa antecessora *j*-1.

O 3sAFS abordado neste trabalho visa encontrar a seqüência de processamento das tarefas que minimizem, simultaneamente, dois objetivos: o tempo total de fluxo e o atraso total. Assim, pretende-se fornecer ao tomador de decisão um conjunto de soluções (programações) eficientes, conhecidas como soluções Pareto-ótimas, de forma tal que o mesmo possa escolher qual delas é a mais adequada para um dado momento.

Na literatura, outras versões do problema de programação em ambientes *assembly flowshop* têm sido estudadas por alguns autores. Algumas abordagens de soluções têm sido propostas para resolver o problema com um único objetivo em dois estágios (2sAFS). Nestes casos, o estágio de transporte é desconsiderado.

Para a minimização do tempo máximo de conclusão (*makespan*) no 2sAFS, algums métodos heurísticos foram desenvolvidos por Potts *et al.* (1995), Hariri e Potts (1997) e Allahverdi e Al-Anzi (2006). Potts et al (1995) provaram que o problema é NP-Difícil e propuseram métodos heurísticos baseados em emuneração implícita. Já em Hariri e Potts (1997) foi apresentado um algoritmo *branch-and-bound* considerando um limitante inferior e critérios de dominância. Allahverdi e Al-Anzi (2006) utilizaram algoritmos evolucionários e, ainda, incorporaram ao problema tempos de preparação não dependentes da sequência.

Para o 2sAFS, com objetivo de minimizar atrasos, há os trabalhos de Al-Anzi e Allahverdi (2007) e Allahverdi e Aydilek (2014). No primeiro, foi proposta a minimização do atraso máximo das tarefas (com tempos de *setup* não dependentes da sequência) através da aplicação de diversas metaheurísticas evolucionárias; enquanto no segundo, abordam-se diversos algoritmos baseados nas metaheurísticas *Simulated Annealing* e Algoritmo Genético, combinados com um algoritmo de inserção (PIA), com intuíto de minimizar o atraso máximo.

O 2sAFS também foi considerado para minimização de vários objetivos pelo uso do método dos pesos (*the weighting method*), onde todas as funções objetivo são combinadas em uma única, linear e ponderada, com a utilização de um vetor de pesos *w*, onde os valores de *w* variam entre 0 e 1. Dessa forma, a abordagem multi-objetivo é reduzida para uma abordagem mono-objetivo. Entre estes trabalhos pode-se destacar os trabalhos de: Allahverdi e Al-Anzi (2008), Torabzadeh e Zandieh (2010), Shokrollahpour *et al.* (2011), Seidgar *et al.* (2013) e Yan *et al.* (2014). Allahverdi e Al-Anzi (2008) propuseram várias heurísticas para minimização do

makespan e tempo médio de fluxo através de três algoritmos heurísticos: *Simulated Annealing* (SA), otimização por colônia de formigas (ACO) e algoritmo de evolução diferencial auto-adaptativo (SDE). Para os mesmos objetivos Torabzadeh e Zandieh (2010) fizeram um *Simulated Annealing* baseado na teoria de nuvem (CSA) e Shokrollahpour *et al.* (2011) propuseram uma algoritmo competitivo imperialista (ICA). Seidgar *et al.* (2013) consideraram um algoritmo competitivo imperialista (ICA) para minimizar o *makespan* e o tempo médio de fluxo. Já Yan *et al.* (2014) propuseram uma variação da metaheurística VNS, combinada com um algoritmo *electromagnetism-like*, para minimização do *makespan* e dos custos de atrasos e adiantamento.

Nós últimos anos, abordagens sobre certas variações do 2sAFS têm sido apresentadas. Fattahi *et al.* (2013) abordaram uma variação do 2sAFS, onde o ambiente de produção no primeiro estágio é um *flowshop* híbrido. Eles propuseram um modelo matemático e várias heurísticas baseadas no algoritmo de *Johnson* para minimização do *makespan*. Já Mozdgir *et al.* (2012) abordaram uma variação do problema que considera múltiplas máquinas não idênticas no estágio de montagem. Eles fazem a minimização da soma ponderada do *makespan* e do tempo médio de conclusão, com tempos de preparação dependentes da sequência no primeiro estágio. Além da resolução através da programação linear mista, eles propuseram uma heurística VNS híbrida para resolução de instâncias maiores. Navaei (2013) também abordou o problema com várias máquinas de montagem não idênticas, porém buscou a minimização do *makespan*, através da aplicação da metaheurística *Simulated Annealing*. Além disso, ele apresentou um caso de estudo real da aplicação do método em uma indústria de manufatura de móveis.

O 3sAFS foi abordado pela primeira vez por Koulamas e Kyparisis (2001). Segundo os autores, este é um modelo mais próxima da realiadade por considerar o estágio de transporte. Os autores analisaram diversas heurísticas construtivas e propuseram a minimização do *makespan*.

Hatami *et al.* (2010) foram os primeiros a tratar o 3sAFS com tempos de *setup* dependentes da sequência das tarefas, porém, aplicado somente às máquinas do primeiro estágio. Eles usaram as metaheurísticas *Simulated Annealing* e Busca Tabu para minimizar a função linear dada pela soma ponderada dos objetivos: atraso máximo e fluxo médio das tarefas. Campos *et al.* (2013) abordaram o mesmo problema e propuseram uma abordagem baseada na combinação das metaheurísticas GRASP e RVND, obtendo melhoria nos resultados finais.

Maleki *et al.* (2012) também utilizaram função linear ponderada com pesos para minimizar tempo médio de conclusão e o *makespan*, através de um algoritmo baseado na metaheurística *Simulated Annealing*. Porém, eles consideraram o 3sAFS com bloqueios. Andrés e Hatami (2011) propuseram um modelo matemático MILP para minimização do *makespan*, considerando tempos de *setup* no primeiro e terceiro estágios.

No melhor do nosso conhecimento, não localizamos na literatura muitos trabalhos que resolvem o problema de programação *assembly flowshop* em uma abordagem de otimização multi-objetivo tradicional, a qual envolve a geração das soluções Pareto-ótimas. Apenas recentemente trabalhos com esse tipo de abordagem têm sido apresentados para o 3sAFS, como por exemplo, os trabalhos de Tajbakhsh *et al.* (2013) e Campos e Arroyo (2014). O primeiro abordou o problema com a utilização de um algoritmo genético e otimização por enxame de partículas (PSO) para minimização do *makespan* e a soma de custos de adiantamentos e atrasos, enquanto o segundo propõe a utilização do tradicional algoritmo para resolução de problemas multi-objectivo NSGA2 (*Non Dominated Sorting Genetic Algorithm* II) combinado com o algoritmo IG (*Iterated Greedy*), para minimização do tempo total de conclusão e atraso total.

Neste trabalho, é feita uma abordagem multi-objetivo tradicional (buscando-se geração ou próximidade de soluções pareto ótimas) para minimização do atraso total e tempo total de fluxo. Aqui é proposto uma melhoria no algoritmo NSGA2_IG desenvolvido por Campos e Arroyo (2014), a partir da adição de uma etapa de busca local em descida, aplicada ao conjunto de soluções não dominadas, em cada iteração.

O NSGA-II (*Elitist Non-dominated Sorting Genetic Algorithm*) proposto por Deb *et al.* (2002) tem sido amplamente utilizado para resolver uma variedade de problemas de otimização multi-objetivos. Já o algoritmo IG, proposto por Ruiz e Stützle (2008), é uma heurística poderosa que tem sido aplicada a todos os tipos de problemas de sequenciamento, obtendo resultados de alta qualidade. A principal característica do IG é a sua simplicidade, que é contrário aos sofisticados algoritmos que incorporam conhecimentos específicos do problema e que, geralmente, têm muitos parâmetros de controle. Apesar de sua simplicidade, o IG tem mostrado resultados *state-of-the-art*, sob diferentes variações de problemas tipo *flowshop* e diferentes objetivos (PAN e RUIZ, 2014).

3.2 Definição do problema

O problema investigado neste trabalho é o sequenciamento de tarefas em um ambiente *Assembly Flowshop* com três estágios. Este problema consiste em processar (ou fabricar) um conjunto de *n* tarefas (produtos), sendo que cada tarefa possui *m* componentes (ou peças) que são fabricados de forma independente. No primeiro estágio, os componentes (peças) das tarefas são fabricados em um ambiente de *m* máquinas paralelas indepedentes e não idênticas. No segundo estágio, as peças produzidas da tarefa são coletadas e transportadas para um local de montagem. No terceiro estágio partes da tarefa são montadas formando o produto final. Os estágios 2 e 3 são constituídos por uma única máquina simples. Todas as máquinas processam apenas uma tarefa por vez, não podem ser interrompidas durante seu processamento e estão disponíveis para serem processadas no tempo zero. Além disso, cada máquina processa apenas uma tarefa por vez.

Observe que para cada tarefa existem *m+2* operações distintas, sendo que *m* operações independentes são feitas no primeiro estágio e as outras duas operações são realizadas nos dois últimos estágios, respectivamente. Os três estágios caracterizam o problema como um ambiente de produção *flowshop*, ou seja, uma tarefa *j* só pode iniciar seu processamento no estágio 2 (transporte) quando todas suas peças são finalizadas no estágio 1, e a montagem (estágio 3) só poderá ser iniciada após a tarefa *j* ser totalmente finalizada no estágio 2.

O sequenciamento das *n* tarefas nas máquinas é representado por uma permutação simples (sequência), que denota a ordem de execução das tarefas nas máquinas e nos estágios, isto é, a sequência das tarefas nos estágios são iguais.

As notações dos dados de entrada do problema são mostradas a seguir:

- n – denota o número de tarefas do problema;
- m – denota o número de máquinas no primeiro estágio;
- i, j – denota as tarefas ($i, j = \{0, 1, 2, \ldots, n\}$, sendo que 0 representa uma tarefa fictícia);
- k – denota uma máquina no primeiro estágio ($k = \{1, 2, \ldots, m\}$);
- $t_{[k,j]}$ – denota o tempo de processamento da tarefa *j* na máquina *k*, no estágio 1;
- $tt_{[j]}$ – denota o tempo de processamento da tarefa *j* na máquina de transporte do estágio 2;

- $ta_{[j]}$ – denota o tempo de processamento da tarefa *j* na máquina de montagem do estágio 3;
- $s1_{[k,j-1,j]}$ – denota o tempo de preparação (*setup*) da máquina *k*, no estágio 1, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1, ..., *n*);
- $s2_{[j-1,j]}$ – denota o tempo de preparação (*setup*) da máquina de transporte, no estágio 2, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1, ..., n);
- $s3_{[j-1,j]}$ – denota o tempo de preparação (*setup*) da máquina de montagem, no estágio 3, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1,..., *n*);
- $d_{[j]}$ – denota a data de entrega da tarefa *j*.

O objetivo do problema consiste em determinar o sequenciamento das tarefas que minimize o tempo total de fluxo (*F*) e o atraso total das tarefas (*T*) simultaneamente, calculado pelas seguintes equações:

$$f_1 = F = \sum_{j=1}^{n} C_{3[j]} \qquad f_2 = T = \sum_{j=1}^{n} max\left\{0,\ C_{3[j]} - d_{[j]}\right\}$$

$C_{3[j]}$, $C_{2[j]}$ e $C_{1[j]}$ representam os tempos de conclusão da tarefa *j*, respectivamente, nos estágios 3, 2 e 1. Eles são calculados pelas seguintes equações:

$$C_{3[j]} = \max\left\{C_{2[j]},\quad C_{3[j-1]}\right\} + s3_{[j-1,j]} + ta_{[j]}$$

$$C_{2[j]} = max\left\{C_{1[j]},\quad C_{2[j-1]}\right\} + s2_{[j-1,j]} + tt_{[j]}$$

$$C_{1[j]} = \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} s1_{[k,i-1,i]} + t_{[j,k]}\right\} + C_{1[j-1]}$$

Os objetivos considerados são de grande importância em sistemas de manufatura. O primeiro objetivo *F* está relacionado à redução do tempo de funcionamento das máquinas e funcionários, ou seja, redução de custos, enquanto o segundo objetivo *T* está relacionado à redução do tempo de armazenamento de produtos (redução de estoques) e atendimento correto dos prazos de entrega, isto é, melhoria na qualidade de atendimento ao cliente. Estes objetivos, para o problema aqui tratado, são de natureza conflitante, ou seja, não há uma solução única *s* que otimize *F* e *T* ao mesmo tempo.

A otimalidade em problemas de otimização multi-objetivo é, portanto, entendida no sentido de Pareto-otimização, isso significa que a resolução de problemas de otimização multi-objetivo consiste em identificar todas as soluções que pertencem ao conjunto Pareto. O conjunto Pareto contém todas as alternativas *s*, não dominadas, por qualquer outra alternativa *s'*. Para comparar corretamente duas soluções em um problema de minimização bi-objetivo, as seguintes definições são utilizadas:

- A solução *s* domina *s′* se o ponto ($f_1(s)$, $f_2(s)$) domina o ponto ($f_1(s')$, $f_2(s')$), tal que, $f_i(s) \leq f_i(s')$ para $i=\{1, 2\}$, e $f_i(s) < f_i(s')$ para pelo menos um *i*.
- A solução *s* é um Pareto-ótimo (ou eficiente) se não existir um *s′*, representado pelo ponto ($f_1(s')$, $f_2(s')$), que domine ($f_1(s)$, $f_2(s)$).

Para exemplificar esta situação, as Figuras 15 e 16 apresentam duas programações (soluções) para uma instância do problema 3sAFS com n=4 tarefas. Os blocos escuros denotam os tempos de preparação necessários à cada máquina. Neste caso, o primeiro estágio tem m=2 máquinas (*M1* e *M2*). As máquinas de transporte e montagem são *Mt* e *Ma*, respectivamente. A data de entrega das tarefas são d_1=20, d_2=15, d_3=43 e d_4=33. Os tempos de processamento das tarefas e os tempos de preparação podem ser vistos nas próprias figuras. Para este exemplo, o conjunto de soluções Pareto-ótimas é formado por quatro sequências de tarefas (soluções): $s1$={2, 1, 4, 3}, $s2$={4, 2, 1, 3}, $s3$={3, 4, 2, 1} e $s4$={4, 3, 2, 1}. Os valores dos objetivos destas sequências são ($F(s1)$, $T(s1)$) = (114, 6), ($F(s2)$, $T(s2)$) = (108, 16), ($F(s3)$, $T(s3)$) = (106, 28) e ($F(s4)$, $T(s4)$) = (104, 29), respectivamente. Pode-se notar que as sequências $s1$ e $s4$ (mostradas respectivamente na Figura 15 e 16) fornecem os valores mínimos para atraso total (T) e o fluxo total (F), respectivamente.

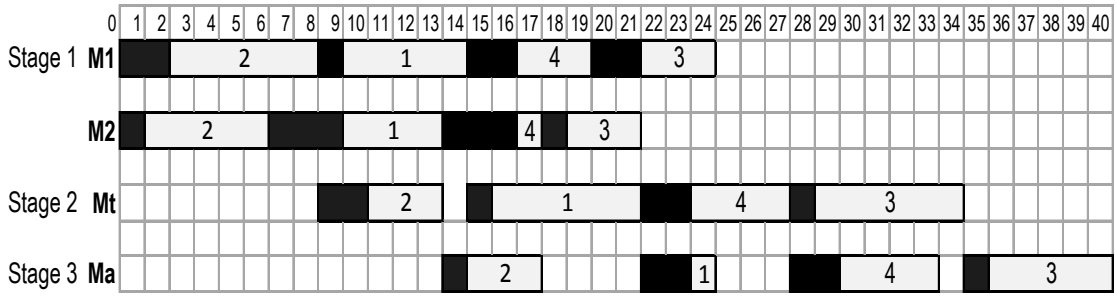

Figura 15: Sequenciamento de tarefas para solução $s1$ = {2, 1, 4, 3} em um exemplo do 3sAFS com 4 tarefas – valores dos objetivos são ($F(s1)$, $T(s1)$) = (114, 6)

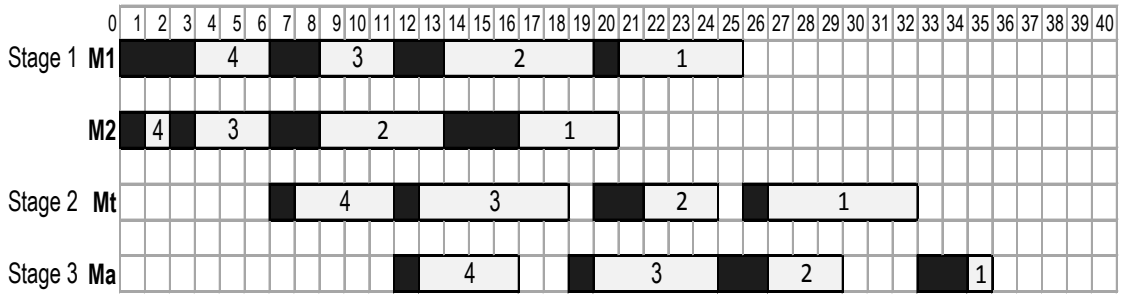

Figura 16: Sequenciamento de tarefas para solução $s4$ = {4, 3, 2, 1} em um exemplo do 3sAFS com 4 tarefas – valores dos objetivos são ($F(s1)$, $T(s1)$) = (104, 29)

A Figura 17 mostra a fronteira Pareto ótima formada para a instância exemplificada. Nota-se que os objectivos F e T estão em conflito, isto é, não há uma única sequência (programação) que minimize F e T ao mesmo tempo. Quando F é reduzido o objetivo T aumenta, e vice-versa.

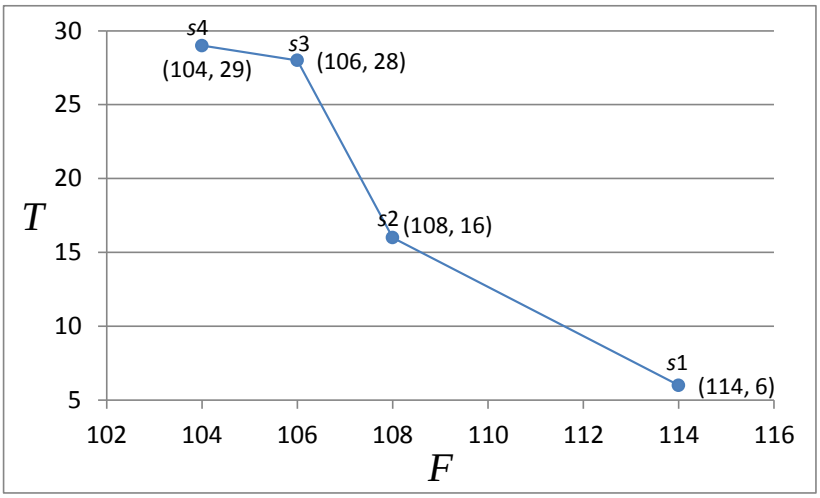

Figura 17: Fronteira Pareto ótimo para o exemplo do 3sAFS com 4 tarefas

Neste trabalho, tem-se por objetivo determinar o conjunto de sequências de tarefas (soluções) Pareto-ótimas, que minimizem F e T, simultaneamente, para o problema 3sAFS bi-objetivo.

3.3 Algoritmos heurísticos propostos

Nesta seção são apresentados os algoritmos heurísticos propostos para resolução do 3sAFS bi-objetivo. Aplicou-se ao problema o algoritmo da literatura NSGA-II (DEB *ET AL.*, 2002) e ainda dois outros algoritmos propostos: o NSGA2_IG, que é uma aplicação combinada do NSGA-II e da metaheurística IG e o NSGA2_IG*, que corresponde a uma melhoria do algoritmo anterior pela adição do passo de busca local em descida.

3.3.1 NSGA-II (*Nondominated Sorting Genetic Algorithm II*)

O NSGA-II (DEB *ET AL.*, 2002) é um algoritmo evolutivo multi-objetivo (algoritmo genético), proposto como uma melhoria do NSGA de Srinivas & Deb (1994). Ele funciona agrupando, por um critério de dominância, as diversas soluções geradas ao longo das iterações do algoritmo, através de um processo denominado "*non-dominated sort*". Nesta abordagem evolucionária as soluções correspondem aos indíviduos da população.

O agrupamento produzido pelo NSGA-II é denominado: "fronteiras". As fronteiras são baseadas no conceito de dominância, como mostrado na Figura 18, onde os pontos (círculos) representam os indivíduos de uma população *P*, agrupados em três fronteiras: F_1, F_2 e F_3. Os indivíduos da fronteira 3 são dominados pelos indivíduos da fronteira 2, que por sua vez, são dominados pelos indivíduos da fronteira 1. Os indivíduos da fronteira 1 dominam todos os outros indivíduos e correspondem ao conjunto de soluções não dominadas (conjunto Pareto-ótimo ou conjunto referência) retornado ao final do algoritmo.

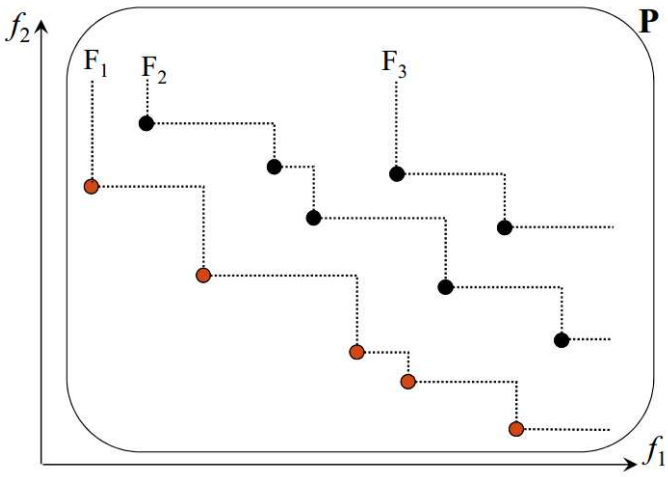

Figura 18: Exemplo do agrupamento por fronteiras feito pelo NSGA-II

Como muitos algoritmos genéticos, o NSGA-II usa uma abordagem elitista, que combina a população anterior com a população gerada na iteração corrente, e, a partir dessa combinação, é extraída uma nova população. Deb *et al*. (2002) ilustram este processo através da Figura 19, exibida a seguir. A Figura mostra que duas populações P_t e Q_t são combinadas formando uma população R_t de tamanho *n*. Após os passos de classificação, combinação e sobrevivência, uma nova população P_{t+1} é gerada.

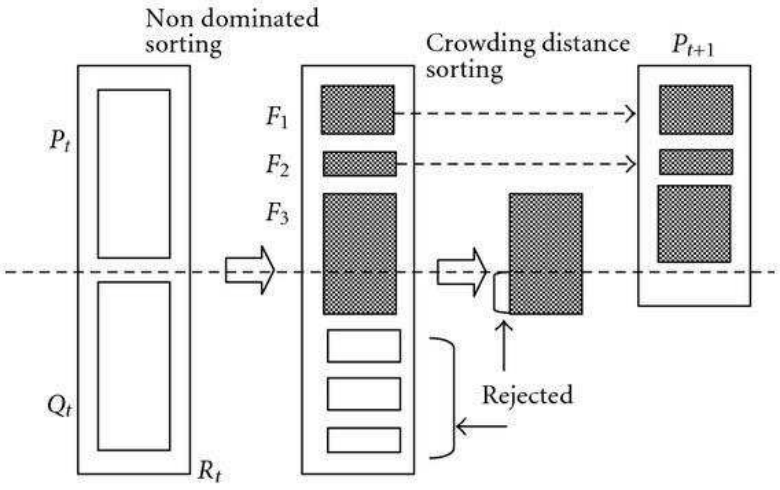

Figura 19: Procedimento NSGA-II (DEB ET AL., 2002)

Para gerar uma nova população, o algoritmo realiza o procedimento de classificação por não-dominância sobre os indivíduos de R_t. Eles são agrupados em fronteiras, sendo que as fronteiras com identificação menor (próximas de zero, como por exemplo, F_1) correspondem a localização dos melhores indivíduos. Em seguida, dentro de cada fronteira, os indivíduos são ordenados de acordo com um operador de diversidade (*Crowding Distance*). Dessa forma, os $^{n}/_{2}$ melhores indivíduos são mantidos para a nova população P_{t+1}, enquanto os restantes, são rejeitados.

Sobre a população elitista formada (P_{t+1}), são realizadas as operações de seleção, cruzamento e mutação, dando origem a uma nova população Q_{t+1}, de tamanho $^{n}/_{2}$. Com as novas populações P_{t+1} e Q_{t+1}, uma nova iteração é iniciada e o processo se repete.

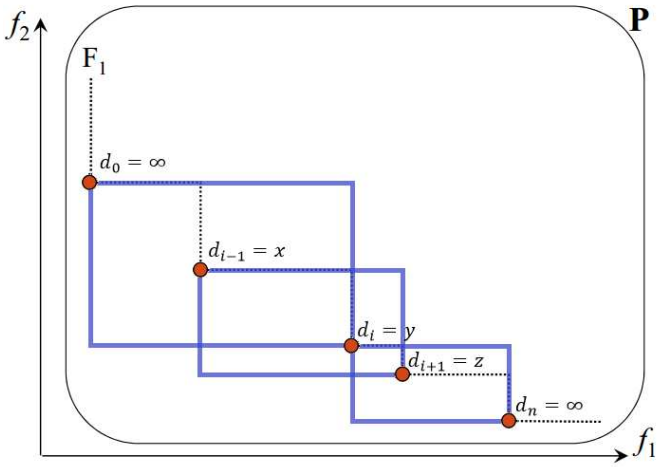

Figura 20: Exemplificação do cálculo da medida "Crowding distance" do NSGA-II

O operador *Crowding Distance* é calculado antes das operações de seleção, cruzamento e mutação. Ele ordena os indivíduos de forma a atribuir precedência àqueles que estiverem em regiões menos povoadas do espaço de objetivos. Cada indivíduo tem um valor de "*crowding distance*" dado pelo perímetro do retângulo formado pelos indivíduos vizinhos, o quais correspondem aos vértices opostos do retângulo, como ilustrado na Figura 20. Nota-se que os pontos extremos da fronteira recebem um valor suficientemente grande (infinito).

O operador de seleção do NSGA-II avalia, além do *fitness* do indíviduo, sua medida de "*crowding distance*" para determinar quais indivíduos farão parte do cruzamento. Em outras palavras, pode-se dizer que o algoritmo prioriza a exploração dos pontos mais isolados da fronteira com o intuito de aumentar a diversidade de soluções (indivíduos) .

O Algoritmo 12 mostra o pseudocódigo do NSGA-II considerado neste trabalho. A única alteração em relação a implementação original é o critério de parada, que neste trabalho é utilizado o tempo máximo de CPU, enquanto na proposta original é sugerido o número máximo de gerações da população. Nesta versão, a primeira fronteira é armazenada no índice 0 (zero) do vetor R_t (vetor que mantém o conjunto de soluções classificadas).

Algoritmo 12: NSGA-II(T , MaxCPU)

```
   Entradas:
     T: Tamanho da população
     MaxCPU: Tempo máximo de CPU
   Saída:
     Dom: Lista de soluções dominantes
   Início
 1 t ← 0; //Indíce de geração da população
 2 Pt ← ∅; //População de T soluções
 3 Qt ← ∅; //População de T soluções descendentes
 4 Rt ← ∅; //Conjunto de soluções classificadas de tamanho 2T
 5 TempoCPU ← 0;
 6 Pt ← GeraPopulacaoInicial(T);
 7 Qt ← GeraPopulacaoInicial(T);
 8 Enquanto (TempoCPU < MaxCPU) faça
 9   Rt ← Pt ∪ Qt;
10   Classificação( Rt ); //Classifica os indivíduos em fronteiras
11   CrowdingDistance( Rt ); //Calcula o "CrowdingDistance" dos indivíduos
12   Pt+1 ← Sobrevivência( Rt ); //Seleciona as T melhores soluções
13   Qt+1 ← GeraFilhos(Pt , T); //Seleção, cruzamento e a mutação
14   t ← t+1;
15   AtualizaTempoCPU(TempoCPU);
16 Fim_Enquanto
17 //Retorna as soluções da fronteira 0 (soluções não dominadas)
18 Dom ← Front( 0, Rt );
19 Retorna Dom;
   Fim
```

No *looping* principal do algoritmo, na linha 9, a população R_t, de tamanho 2*T*, recebe os indivíduos das duas populações *P* e *Q*. O método "*Classificação*" (linha 10) aplica o processo "*non-dominated sort*" classificando os indivíduos de R_t em fronteiras. O método "*CrowdingDistance*" (linha 11) ordena os indivíduos de cada fronteira de R_t de acordo com a medida *CrowdingDistance*. O método "*Sobrevivência*" (linha 12) atribui a população P_{t+1} os *T* melhores indivíduos de R_t. O método "GeraFilhos" (linha 13) aplica o operador de seleção, cruzamento e mutação, comum aos algoritmos de natureza evolutiva.

Após a aplicação deste métodos, é realizada a atualização do indíce da geração da população e o tempo de CPU (linhas 14 e 15). O algoritmo retorna ao final a fronteira 0 da população R_t, que corresponde ao conjunto de soluções não dominadas, através do método "*Front*" (linha 18 e 19).

As próximas seções explicam o funcionamento do método de geração de soluções iniciais para definição da população zero e também dos operadores de seleção, cruzamento e mutação.

3.3.1.1 Geração da solução inicial

O algoritmo "*GeraPopulacaoInicial*" do NSGA-II produz uma população com 2*T* soluções, sendo *T* indivíduos distribuídos na população *P* e a outra metade na população *Q*, conforme visto nas linhas 5 e 6 do Algoritmo 12. A primeira parte desta população é gerada pela heurística multi-objetivo parcial enumeração (MOPE), proposta por Arroyo e Armentano (2004). A outra parte da população é gerada aleatoriamente.

A heurística MOPE é inspirada nas heurísticas NEH propostas por Nawaz *et al.* (1984). A heurística MOPE utiliza o conceito de dominância de Pareto para manter não apenas uma solução parcial incompleta em cada iteração (como na NEH), mas todo um conjunto de soluções não-dominadas parciais geradas durante o processo de inserção das tarefas. Para minimizar a combinação de objetivos, tempo total de fluxo e atraso total, a heurística MOPE usa as seguintes regras de despacho:

- SPT (*Shortest Processing Time*): as tarefas são organizadas em ordem ascendente do tempo total de processamento, calculado pela seguinte equação:

$$SPT_{[j]} = \left(\max_{k=1,\dots,m} t_{[k][j]}\right) + tt_{[j]} + ta_{[j]} \quad \forall j = \{1, \dots, n\}$$

- TLB (*Tardiness Lower Bound*): as tarefas são organizadas em ordem descendente de acordo com o limitante inferior de atrasos de cada tarefa, calculado pela seguinte maneira:

$$TLB_{[j]} = d_{[j]} - \left(\left(\max_{k=1,\dots,m} t_{[k][j]} \right) + tt_{[j]} + ta_{[j]} \right) \quad \forall j = \{1, \dots, n\}$$

A heurística MOPE gera T_h (<2T) soluções não-dominadas. Então, 2T–T_h soluções são geradas aleatoriamente.

3.3.1.2 Seleção

Para criar uma população descendente (filhos) Q_t, os seguintes operadores são executados: seleção, cruzamento e mutação.

A partir da seleção de duas soluções base (soluções pai), duas novas soluções filhas são criadas. Cada solução pai (indivíduo) é escolhida na população P_t usando a técnica torneio binário. Esta técnica consiste em escolher aleatoriamente duas soluções da população e o melhor é selecionado. As soluções que pertencem às fronteiras menores (próximas de zero) são sempre as melhores. Ao comparar duas soluções pertencentes a mesma fronteira, o valor atribuído de "*crowding distance*" é observado, então o indivíduo com a maior medida de "*crowding distance*" é considerado vencedor.

3.3.1.3 Cruzamento

O operador de cruzamento (*crossover*) cria soluções (indivíduos) descendentes por coalescência de duas soluções pai. O objetivo é gerar melhores filhos, ou seja, gerar soluções com melhor qualidade após a realização do método de cruzamento. Muitos diferentes operadores de cruzamento têm sido propostos na literatura para problemas de sequenciamento *flowshop*. Neste trabalho, usamos três operadores descritos na literatura:

- *Similar Block 2-Point Order Crossover* – SB2OX (RUIZ e MAROTO, 2005);
- *Two-Point Crossover* – TPX (ISHIBUCHI *ET AL.*, 2003);
- *Partially matched crossover* – PMX (SIVANANDAM e DEEPA, 2008).

Todos os operadores são baseados em um ponto de corte aplicado à sequência de tarefas das soluções pai. O início i do ponto de corte na sequência e o seu tamanho p são escolhidos aleatoriamente, de acordo com os respectivos intervalos: $i \in \{1, \dots, n\}$ e $p \in \{1, \dots, n - i\}$. Além disso, o cruzamento é realizado de acordo com uma probabilidade.

O operador SB2OX avalia todas as posições da sequência em ambos os pais e identifica as tarefas idênticas que estão na mesma posição em ambas as sequências; logo após, persiste para os filhos essas tais tarefas para as mesmas posições. Em seguida, é transferido para o filho 1 o ponto de corte do pai 1, e, de forma similar, o pai 2 transfere para o filho 2. Por fim, as lacunas da sequência do filho 1 são completadas pelas tarefas faltantes, obedecendo a mesma ordem em que elas estão arranjadas no pai 2; similarmente, o mesmo ocorre para o filho 2, em relação ao pai 1.

A Figura 21 ilustra este procedimento com base nos pais *P1* e *P2*, onde é considerado um ponto de corte iniciado na posição i=3 da sequência e com tamanho p=4. O cruzamento dá origem as soluções filhas *F1* e *F2*.

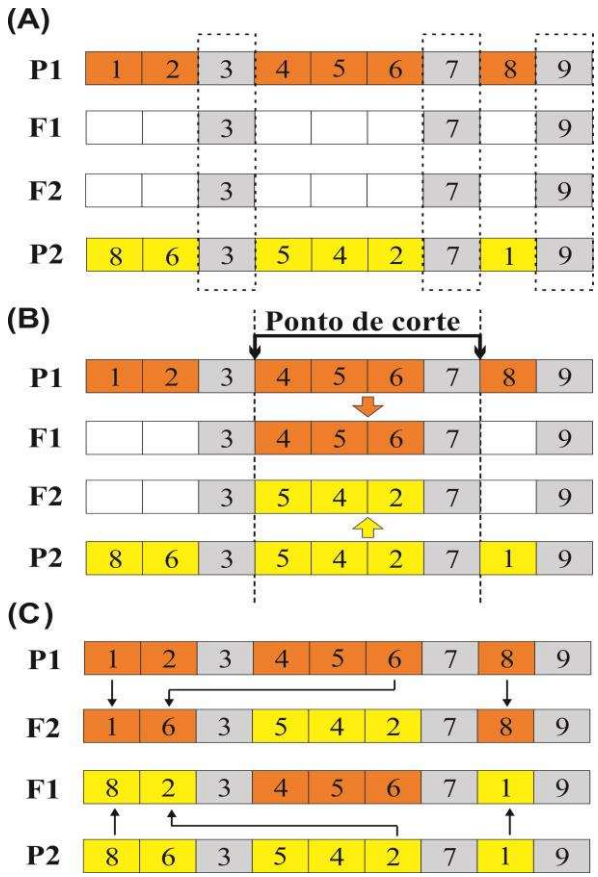

Figura 21: Cruzamento pelo operador SB20X – (a) em primeiro lugar, as tarefas comuns em ambos os pais são copiadas para os filhos (cor cinza); (b) em seguida, as tarefas do ponto de corte são herdadas do pai para os filhos (c) as tarefas faltantes são copiadas em ordem do outro progenitor.

O operador PTX funciona copiando um bloco de tarefas, resultante da aplicação do ponto de corte na sequência do pai 1, para o filho 1, e, também, copiando para o filho 2, os blocos de tarefas externos ao ponto de corte aplicado à sequência de tarefas do pai 2. Logo após, as lacunas da sequência do filho 1 são completadas pelas tarefas faltantes, obecendo a ordem

definida pela sequência de tarefas do pai 2. De maneira similar, o mesmo ocorre para o filho 2, em relação ao pai 1.

A Figura 22 ilustra este procedimento com base nos pais *P1* e *P2*. É considerado um ponto de corte iniciado na posição i=3 da sequência, com tamanho p=4. O cruzamento dá origem aos filhos *F1* e *F2*.

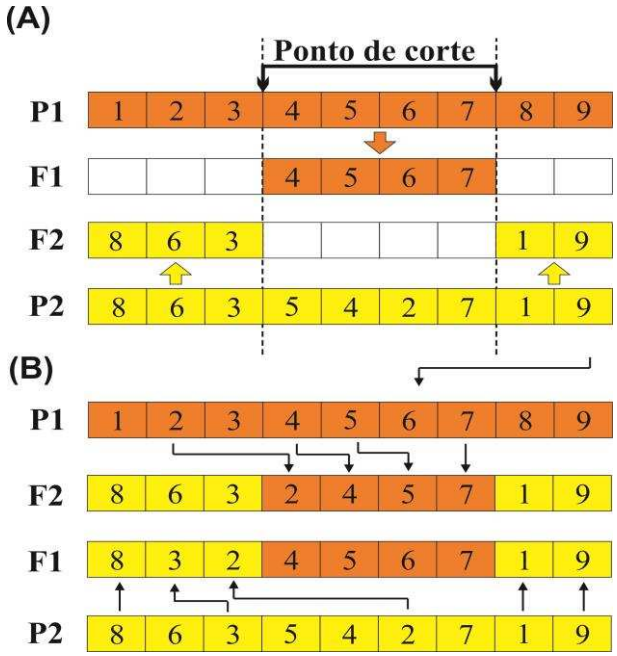

Figura 22: Cruzamento pelo operador PTX – (a) em primeiro lugar, o bloco de tarefas correspondente ao ponto de corte são herdadas do pai 1 para o filho 1, enquanto o filho 2 herda os blocos de tarefas externo ao ponto de corte do pai 2, (b) em seguida, as tarefas faltantes são copiadas em ordem do outro progenitor.

O operador PMX funciona similarmente ao SB20X. Porém, não copia as tarefas que ocupam as mesmas posições no pai para os filhos. O operador simplesmente copia o bloco de tarefas definida pelo ponto de corte do pai 1, para o filho 1, e, da mesma maneira, o pai 2 transfere para o filho 2 o bloco correspondente ao seu ponto de corte da sua sequência.

A Figura 23 ilustra este procedimento com base nos pais *P1* e *P2*. É considerado um ponto de corte iniciado na posição i=2 da sequência, com tamanho p=4. O cruzamento dá origem as soluções filhas *F1* e *F2*.

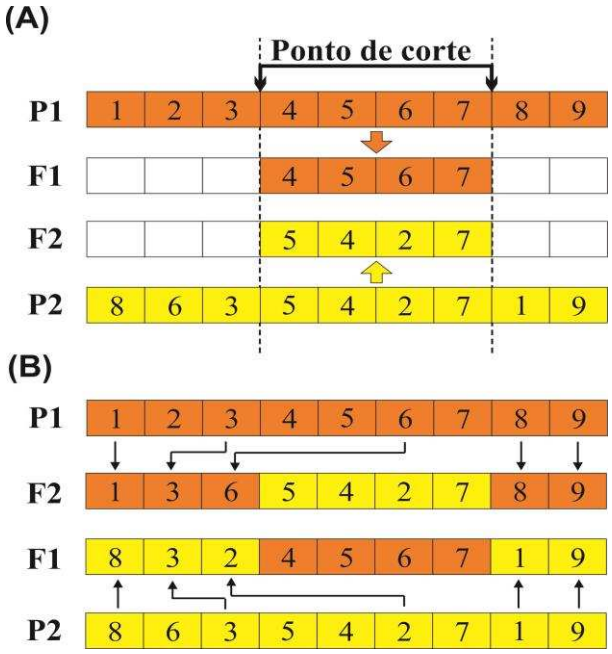

Figura 23: Cruzamento pelo operador PMX – (a) o bloco de tarefas correspondente ao ponto de corte são herdadas do pai 1 para o filho 1, enquanto o filho 2 herda o bloco de tarefas definido pelo ponto de corte no pai 2, (b) em seguida, as tarefas faltantes são copiadas na ordem do outro progenitor.

3.3.1.4 Mutação

O operador de mutação aqui usado é o operador de troca. Este operador escolhe, aleatoriamente, duas posições diferentes dentro da sequência de tarefas e faz *swap* dessas tarefas. A mutação ocorre de acordo com uma probabilidade de mutação. A cada vez que o procedimento de mutação é realizado sobre uma solução filha, duas trocas (*swap*) são executadas.

3.3.2 NSGA2_IG

O algoritmo NSGA2_IG é a aplicação combianda do NSGA-II padrão e da metaheurística *Iterated Greedy* (IG). O IG é usado para acelerar a velocidade de convergência do algoritmo NSGA-II, através de um processo de intensificação gulosa, nas soluções contidas na primeira fronteira (local onde encontra-se as soluções não-dominadas).

O IG é um método simples que tem sido aplicado a diferentes tipos de problemas de sequenciamento de tarefas mono-objetivo. A primeira aplicação do IG para problemas de programação *flowshop* foi feita por Ruiz e Stützle (Ruiz e Stützle, 2008). Minella *et al.* (2011) propõe uma adaptação do IG para problemas de programação *flowshop* multi-objetivo. Arroyo *et al.* (2011) usou o mesmo procedimento de intensificação em uma heurística multi-objetivo VNS para resolver um problema de programação em uma única máquina.

A Figura 24 mostra a ideia básica de uma iteração do NSGA2_IG. Nota-se que uma população *D*' é gerada a partir da população P_t, pela aplicação do IG, e, em seguida, é inserida na população R_t. A partir de então, os passos básicos do NSGA-II (classificação, "*crowding distance*", sobrevivência, seleção, cruzamento e mutação) são aplicados normalmente.

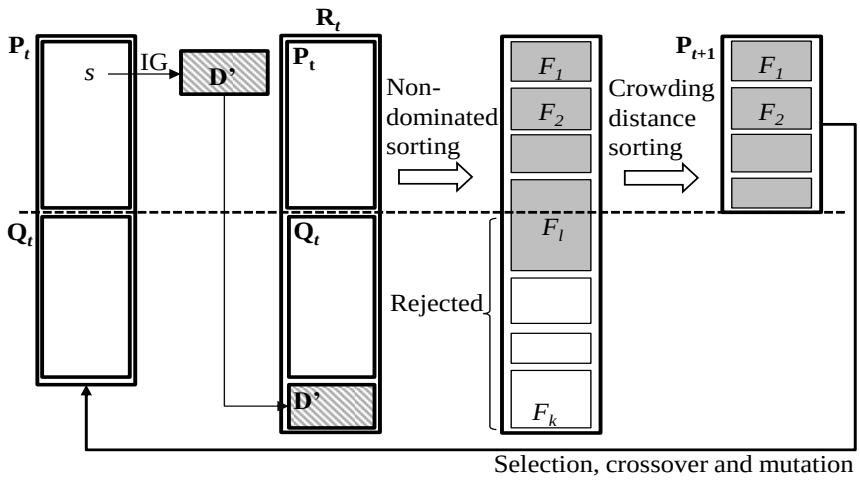

Figura 24: Processo básico do NSGA2_IG.

O pseudocódigo do NSGA2_IG está descrito no Algoritmo 13.

Algoritmo 13: NSGA2_IG(*T* , *d*, *MaxCPU*)

```
Entradas:
   T: Tamanho da população
   d: Parâmetro usado na estratégia do IG
   MaxCPU: Tempo máximo de CPU
Saída:
   Dom: Lista de soluções dominantes
Início
 1 t ← 0; //Indíce de geração da população
 2 Pt ← ∅; //População de T soluções
 3 Qt ← ∅; //População de T soluções descendentes
 4 Rt ← ∅; //Conjunto de soluções classificadas em fronteiras
 5 D' ← ∅; //Conjunto de soluções não dominadas geradas pelo IG
 6 TempoCPU ← 0;
 7 Pt ← GeraPopulacaoInicial(T);
 8 Qt ← GeraPopulacaoInicial(T);
 9 Enquanto (TempoCPU < MaxCPU) faça
10   Se (t ≥ 1) Então
11     S ← SolucaoRandom( 0, Pt );//Pega uma solução randômica da fronteira 0
12     D' ← IG( S, d );
13   Rt ← Pt ∪ Qt ∪ D';
14   Classificação( Rt ); //Classifica os indivíduos em fronteiras
15   CrowdingDistance( Rt ); //Calcula o "CrowdingDistance" dos indivíduos
16   Pt+1 ← Sobrevivência( Rt ); //Seleciona as T melhores soluções
17   Qt+1 ← GeraFilhos(Pt , T); //Seleção, cruzamento e mutação
18   t ← t+1;
19   AtualizaTempoCPU(TempoCPU);
20 Fim_Enquanto
21 //Retorna as soluções da fronteira 0 (soluções não dominadas)
22 Dom ← Front( 0, Rt );
23 Retorna Dom;
Fim
```

O Algoritmo 13 recebe como entradas os parâmetros *T*, *d* e *MaxCPU*. O parâmetro *T* determina o tamanho da população, já o parâmetro *d* é utilizado pelo procedimento IG e o parâmetro *MaxCPU* é o critério de parada (tempo máximo de execução na CPU).

Inicialmente (iteração t=0), duas populações de indivíduos (soluções) P_t e Q_t, cada uma de tamanho *T*, são criadas, conforme explicado na seção 3.3.1.1 (linhas 6 e 7 do Algoritmo 13). Dentro do *looping* principal do algoritmo, a partir da segunda iteração ($t \geq 1$), é executado o procedimento IG (que corresponde a metaheurística *Iterated Greedy*). O procedimento IG recebe como entrada uma solução não dominada *S*, selecionada aleatoriamente da fronteira F_0 de P_t. O IG aplica sobre a solução *S* o tradicional método de destruição-construção (DC), que age como uma busca local gulosa (este procedimento é explicado com mais detalhes na próxima seção). O conjunto de soluções não dominadas resultante deste processo é retornado para a população *D'*.

As três populações de P_t, Q_t e D' são combinadas para formar uma população R_t de tamanho $2T + |D'|$, onde $|P_t| = |Q_t| = T$. Em seguida, é realizada a classificação por dominância ("*nom-dominated sort*") para geração das fronteiras Pareto sobre os indíviduos da população R_t, sendo $R_t = P_t \cup Q_t \cup D'$. Os próximos passos da iteração são os mesmos já mencionados e também realizados pelo NSGA-II: cálculo do *Crowding distance*, sobrêvivencia, seleção, cruzamento e mutação.

O procedimento IG é executado a cada iteração (exceto na primeira) e realiza um procedimento de intensificação gulosa sobre uma solução não dominada da fronteira 0, gerando novas soluções de boa qualidade. A próxima subseção descreve o tal procedimento.

3.3.2.1 Intensificação gulosa com o IG

O procedimento IG é composto de duas fases: destruição e construção. O pseudocódigo deste procedimento é mostrado no Algoritmo 14. Ele recebe como entradas uma solução *S* e o número *d* de tarefas que serão desconectadas da sequência durante a fase de destruição.

Na fase de destruição, *d* tarefas são selecionadas aleatoriamente da sequência *s* e removidas, criando assim uma solução s_p parcial de tamanho *n–d* (linhas 3, 4 e 5 do Algoritmo 14). As tarefas removidas são armazenadas em um vetor *J* (onde $J_{[i]}$, i=1,...,*d*, são tarefas removidas).

```
Algoritmo 14: IG (S, d)
  Entradas:
     s: Solução escolhida aleatoriamente de F0
     d: Número de tarefas que serão removidas da sequência
  Saída:
     D': Conjunto de soluções não dominadas obtidas de S
  Início
 1   J  ← ∅
 2   //Fase de destruição da sequência:
 3   Para (i ← 1 até d) faça
 4     J[i] ←remove uma tarefa selecionada aleatoriamente de S;
 5   sp ← s; //solução parcial com n-d tarefas
 6   //Fase de Construção:
 7   D' ← {sp}
 8   Para ( i ← 1 to d) faça
 9     Di ← ∅; //armazena as soluções não dominadas da iteração
10     Para cada (solução parcial s' de D') faça
11       Ds' ← soluções não dominadas obtidas após a inserção
               da tarefa J[i] em todas as posições de s';
12       Di ← soluções não dominadas de Di ∪ Ds';
13     Fim_Para
14     D' ← Di;
15   Fim_Para
16   Retorna D';
  Fim
```

Já a fase de construção tem d passos. No passo i=1, n–d+1 soluções parciais são construídas através da inserção da tarefa $J_{[1]}$ em todas as posições possíveis de s_p. A partir das n–d+1 soluções parciais, um conjunto D_i de soluções parciais não dominadas é formado. No passo seguinte, as novas soluções (de tamanho n–d+2) são obtidas pela inserção da tarefa $J_{[2]}$ em cada uma das outras soluções parciais não dominadas. De forma iterativa, a cada passo, é realizada a construção de novas soluções parcias e as soluções não dominadas são sempre selecionadas e armazenadas. Por fim, no passo d, um conjunto D' de soluções completas e não-dominadas é formado.

A Figura 25 ilustra a idéia do procedimento de intensificação gulosa proposto pelo procedimento IG. No exemplo, ele começa a partir de uma solução s com n=4 tarefas e é considerado d=2 para remoção das tarefas. São removidas da sequência as tarefas 2 e 4 (respectivamente). A partir da inserção da tarefa 2, são geradas as sequências parciais {2,1,3}, {1,2,3} e {1,3,2}. As sequências em verde correspondem as soluções parciais não dominadas. As soluções não dominadas são armazenadas e na iteração seguinte a tarefa 4 é reinserida, produzindo novas soluções, porém, neste momento as sequências (soluções) já estão completas. As soluções não dominadas (em verde) {4,2,1,3}, {2,1,3,4}, {1,4,3,2} e {1,3,2,4} são selecionadas e retornadas pelo procedimento IG.

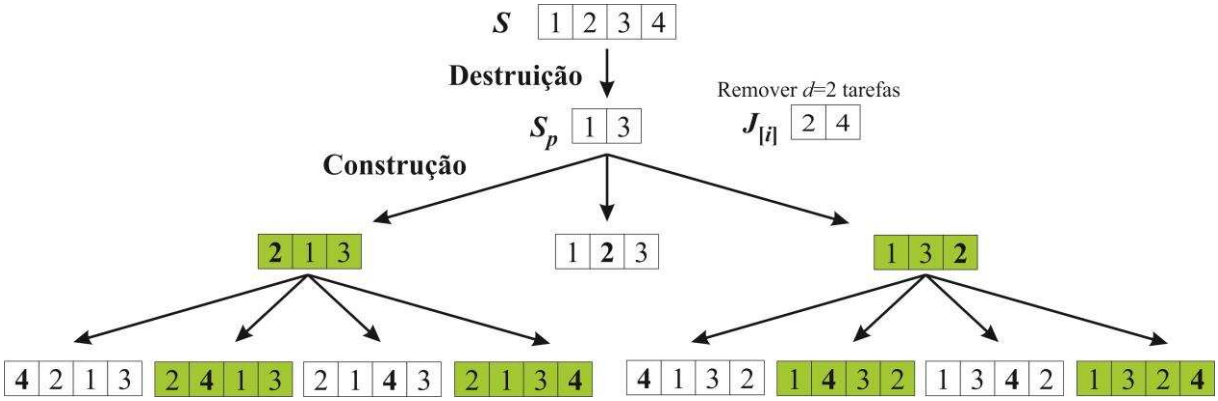

Figura 25: Procedimento de intensificação Iterated Greedy (IG).
As sequências em verde correspondem as soluções não-dominadas.

3.3.3 NSGA2_IG*

O algoritmo NSGA2_IG* pode ser considerado uma versão melhorada do NSGA2_IG. Nesta versão, as soluções não dominadas da primeira fronteira são ainda mais exploradas, pois o método IG é aplicado em várias diferentes soluções em uma mesma iteração (e não apenas em uma). Além disso, é acrescido um processo de busca local em descida, aplicado sobre as soluções retornadas pelo processo de destruição-construção do IG*. O Algoritmo 15 mostra o pseudocódigo do NSGA2_IG*.

Algoritmo 15: NSGA2_IG*(T , MaxCPU)

```
   Entradas:
      T: Tamanho da população
      MaxCPU: Tempo máximo de CPU
   Saída:
      Dom: Lista de soluções dominantes
   Início
 1 t ← 0; //Indíce de geração da população
 2 Pt ← ∅; //População de T soluções
 3 Qt ← ∅; //População de T soluções descendentes
 4 Rt ← ∅; //Conjunto de soluções classificadas em fronteiras
 5 D' ← ∅; //Conjunto de soluções não dominadas geradas pelo IG
 6 TempoCPU ← 0;
 7 Pt ← GeraPopulacaoInicial(T);
 8 Qt ← GeraPopulacaoInicial(T);
 9 Enquanto (TempoCPU < MaxCPU) faça
10   Se (t ≥ 1) Então
11     D' ← IG*( Front(0, Pt) );
12   Rt ← Pt ∪ Qt ∪ D';
13   Classificação( Rt ); //Classifica os indivíduos em fronteiras
14   CrowdingDistance( Rt ); //Calcula o "CrowdingDistance" dos indivíduos
15   Pt+1 ← Sobrevivência( Rt ); //Seleciona as T melhores soluções
16   Qt+1 ← GeraFilhos(Pt , T); //Seleção, cruzamento e mutação
17   t ← t+1;
18   AtualizaTempoCPU(TempoCPU);
19 Fim_Enquanto
20 //Retorna as soluções da fronteira 0 (soluções não dominadas)
21 Dom ← Front( 0, Rt );
22 Retorna Dom;
   Fim
```

É possível perceber que a única diferença entre o pseudocódigo apresentado no Algoritmo 15 e o apresentado no Algoritmo 13 (NSGA2_IG), está na linha 11: Não é feita a seleção aleatória de uma solução da fronteira F_0 antes da aplicação do IG. Além disso, aqui o procedimento é chamado de IG*, indicando que trata-se de um procedimento diferente.

O Algoritmo 16 mostra o pseudocódigo do IG*. Ele recebe como entradas, o parâmetro *F*, que corresponde ao conjunto de soluções não dominadas (toda as soluções contidas na fronteira F_0), o parâmetro *q* ($0 < q \leq 1$), que corresponde ao percentual de soluções de *F* que serão exploradas, e o parâmetro *p*, que corresponde a probabilidade de aplicação da busca local em descida em uma dada solução *s*.

```
Algoritmo 16: IG*(F, q, p)
  Entradas:
     F: Conjunto de soluções não dominadas da fronteira F0
     q: Percentual de soluções de F que são exploradas
     p: Constante de probabilidade de aplicação da busca local
  Saída:
     R: Conjunto de soluções não dominadas obtidas de F
  Início
 1   R ← ∅;
 2   nSol ← Tamanho(F)*q; //número de soluções exploradas pelo IG*
 3   Para (i ← 1 até nSol) faça
 4     s ← SolucaoRandom( F ); //Pega uma solução randômica de F
 5     d ← Random( 2, 6 ); //seleciona número aleatório entre 2 e 6
 6     C ← DestruicaoConstrucao(s, d); //aplica DC multi-objetivo
 7     //aplicação da busca local sobre as soluções de C
 8     Para cada (solução c' de C) faça
 9       R ← Dominantes( R ∪ c');//c' é inserido em B caso seja dominante
10       Se ( Random( 1, 100 ) ≤ p ) então
11         W ← BuscaLocalMult( c', R, y );//y parâmetro da busca local
12         R ← Dominantes( R ∪ W );
13     Fim_Para
14   Fim_Para
15   Retorna R;
  Fim
```

Através do Algoritmo 16 é possível ver o funcionamento do IG* . Ele inicaliza um conjunto de soluções *R* para armazenar as soluções não dominadas que serão retornados ao final. Em seguida, é aplicado o processo de destruição-construção em *nSol* soluções de *F*, escolhidas aleatoriamente. O valor de *nSol* é determinado pelo parâmetro *q*, na linha 2. Em seguida, o procedimento "*DestruicaoConstrucao*" é executado de maineira similar ao processamento descrito na seção 3.3.2.1, e, as soluções não dominadas resultante são armazenadas no conjunto *C*. O próximo passo é a aplicação da busca local (iniciada na linha 7). Para cada uma das soluções *c'* de *C* é aplicada uma busca local em descida, multi-objetivo,

de acordo com um critério de probabilidade definido na linha 9. Se alguma solução *c'* de *C* é dominante, ela é inserida diretamente no conjunto *R* (linha 8), sem a aplicação da busca local. O procedimento "*BuscaLocalMult*" é o método responsável por executar a busca local em descida. Ele é aplicado na sequência e devolve como resultado um conjunto de soluções vizinhas não dominadas, que são adicionadas ao conjunto *R* (linha 11). O processo continua iterativamente até que as *nSol* soluções de *F* tenham sido exploradas. Ao final, têm-se em *R* um conjunto de soluções não dominadas. R é atribuído ao conjunto *D'* do algoritmo principal (NSGA2_IG*).

Há uma tênue diferença entre o processamento de destrução-construção realizado pelo método IG do NSGA2_IG, descrito na seção 3.3.2.1, e o processamento aqui realizado pelo IG*. Enquanto o IG remove *d* tarefas aleatórias da sequência, sendo *d* uma constante fixa, o IG* remove um bloco completo de *d* tarefas adjacentes, sendo que *d* corresponde ao tamanho do bloco e é determinado aleatoriamente, de forma que $d \in [2, 6]$.

A próxima seção explica em detalhes o processamento executado pela busca local em descida "*BuscaLocalMult*" e a estrutura de vizinhança utilizada.

3.3.3.1 Busca local em descida

O Algoritmo 17 mostra a implementação do processo de busca local em descida.

```
Algoritmo 17: BuscaLocalMult(s, R, y)
  Entradas:
     s: Solução base para busca local
     R: Conjunto corrente de soluções não dominadas
     y: número de tarefas que serão movimentadas na sequência s
  Saída:
     B: Conjunto de soluções não dominadas por R
  Início
 1   B ← R;
 2   k ← n/y; //n corresponde ao tamanho da sequência s
 3   z ← Random(4, 10);
 4   V ← Vizinhanca( s, k, z ); //Gera k*z*2 soluções vizinhas
 5   D ← Dominantes( V ); //Retorna apenas as soluções não dominadas de V
 6   Para cada (solução d de D) faça
 7     Se ( NãoDominado( d, B ) ) então
 8       B ← Dominantes( d ∪ B );
 9       B ← BuscaLocalMult( d, B, y );
10   Fim_Para
11   Retorna B;
  Fim
```

O algoritmo "*BuscaLocalMult*" recebe como entrada uma solução *s*, onde será aplicada a busca, o conjunto corrente de soluções não dominadas *R* e um parâmetro que define a quantidade de tarefas que serão movimentadas no processo de geração da vizinhança. O conjunto R é necessário ao processo para avaliar se um vizinho gerado é dominante ou não, pois a "descida" é realizada somente em vizinhos que não são dominados por outras soluções.

Como se pode ver no algoritmo 17, o "*BuscaLocalMult*" retorna um conjunto de soluções não dominadas *B*. Inicialmente *B* é igual a *R* (linha 1), pois caso não seja encontrada mais nenhuma solução não dominada o conjunto de soluções já conhecido é retornado. As linhas 2 e 3 definem o valor dos parâmetros *k* e *z* necessários para geração dos vizinhos. Na linha 4 os vizinhos da solução *s* são gerados e armazenados em *V*. O conjunto *D* recebe apenas as soluções não dominadas do conjunto *V* (linha 5). Em seguida, as *d* soluções de *D* tem sua dominância avaliada em relação ao conjunto *B*, na linha 7. O método "*NãoDominado*" retorna verdadeiro caso não exista solução em *B* que domine *d*, caso contrário, é retornado falso. Então, se *d* é uma solução não dominante, ela é armazenada em *B* (linha 8) e uma chamada recursiva para o algoritmo é realizada, passando como parâmetros a solução *d* que está sendo explorada e o conjunto de todas as soluções não dominadas *B*. Assim, todo processo irá se repetir, porém em uma nova instância do procedimento *BuscaLocalMult*. As soluções não dominadas encontradas nas outras instâncias do *BuscaLocalMult* também são adicionadas ao conjunto *B* (linha 9) e, ao final, têm-se um conjunto com todas as soluções não dominadas conhecidas.

A variável *k* é definida em função do parâmetro de entrada *y* e da quantidade de tarefas *n* do problema considerado, ou seja, se para uma dada instância n=30 e y=6, o valor de *k* será 5. Já o parâmetro *z* é definido aleatoriamente de acordo com o intervalo [4, 10]. O procedimento "*Vizinhança*" gera $k \times z \times 2$ vizinhos (conforme explicado na seção seguinte).

A Figura 26 ilustra um exemplo da "descida" realizada pela *BuscaLocalMult*.

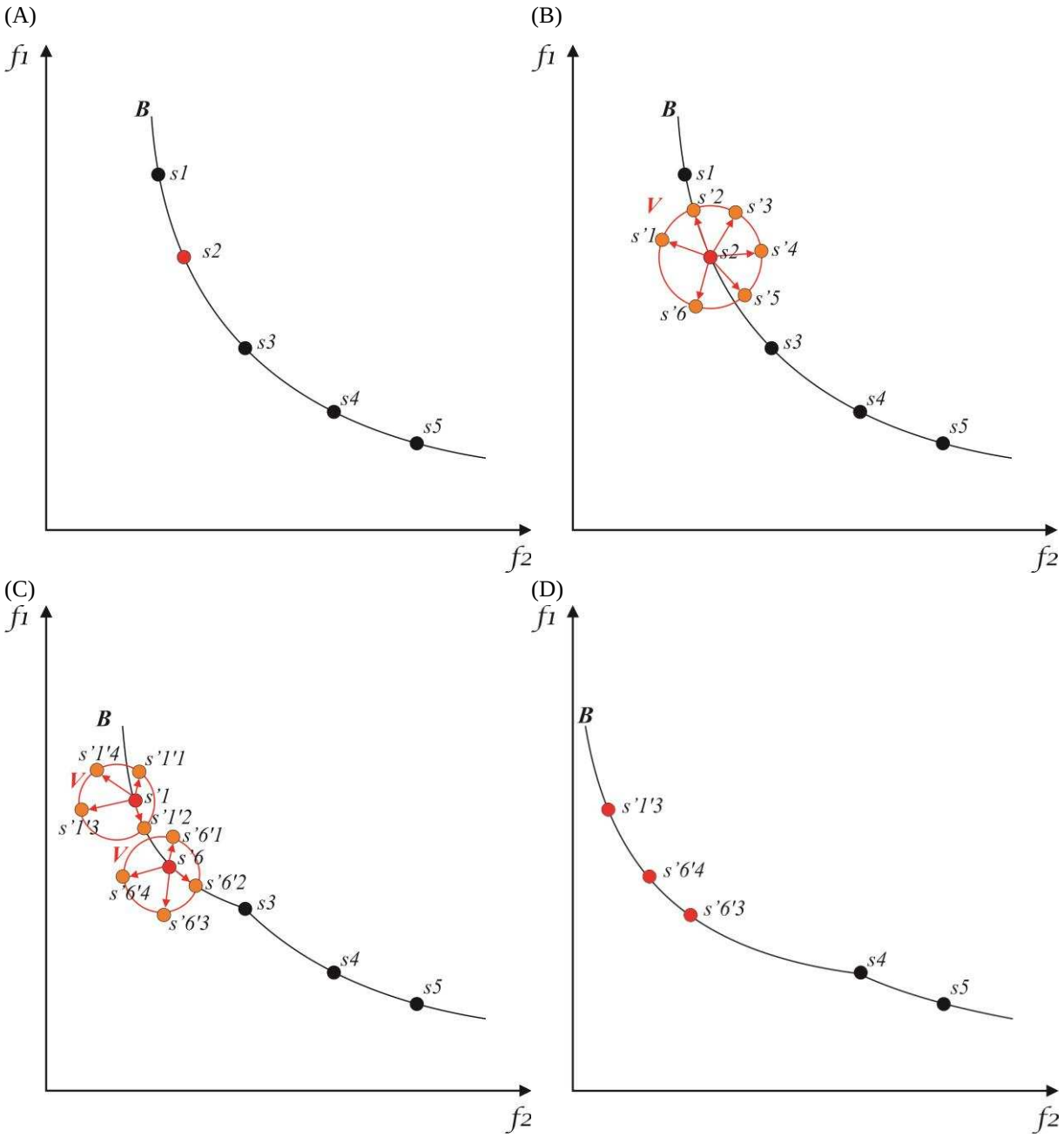

Figura 26: Exemplo da busca local em descida – (A) mostra o conjunto de soluções não dominadas *B*; (B) mostra a geração de vizinhos a partir da solução *s2*, os vizinhos *s'1* e *s'6* dominam as soluções *s1* e *s2* e são incluídos em *B*; (C) a busca continua sobre as novas soluções *s'1* e *s'2*, gerando novas soluções dominantes; (D) Conjunto *B* resultante do final do processo de busca local

Observa-se pela Figura 26 que, em (A) o procedimento é iniciado com o conjunto *B*, formado pelas soluções não dominadas *s1*, *s2*, *s3*, *s4* e *s5*. A solução *s2* (em vermelho) é a solução que sofrerá a busca local. Em (B) são mostrados os vizinhos *V* gerados a partir de *s2*: *s'1*, *s'2*, *s'3*, *s'4*, *s'5* e *s'6*. Os vizinhos *s'3*, *s'4* e *s'5* são dominados pelas outras soluções e por isso serão descartados de *B*, porém a solução *s'1* domina as soluções *s1* e *s'2*, enquanto a solução *s'6*, domina a solução *s2*. Tanto *s'1* quanto *s'6*, não são dominadas por outras soluções, por isso serão incorporadas ao conjunto *B*. Em (C) repare-se que as soluções dominadas já

foram removidas e as não dominadas já foram incluídas, gerando um novo conjunto *B* = {*s'1*, *s'6*, *s3*, *s4*, *s5*}. Ainda em (C), percebe-se que as novas soluções incluídas são exploradas e novos vizinhos são gerados a partir delas. Este fato é o que constitui o processo de "descida" da busca local. A solução *s'1* dá origem aos vizinhos *s'1'1*, *s'1'2*, *s'1'3* e *s'1'4*, enquanto *s'6* origina os vizinhos *s'6'1*, *s'6'2*, *s'6'3* e *s'6'4*. É notório que os vizinhos *s'1'3*, *s'6'3* e *s'6'4* dominam os outros vizinhos gerados e também as soluções *s'1*, *s'6* e *s3*. Assim, em (D), novamente o conjunto *B* é atualizado, incluindo as soluções não dominadas e retirando as que foram dominadas. Conforme pode ser visto (D), as soluções (em vermelho) *s'1'3*, *s'6'3* e *s'6'4* devem ser exploradas, seguindo a mesma ideia aqui apresentada, porém, supondo que não são gerados vizinhos melhores que as soluções já conhecidas em *B*, é retornado como resultado as seguintes soluções não dominadas *B*={ *s'1'3*, *s'6'3*, *s'6'4*, *s4* e *s5*}.

Na próxima seção é detalhada a forma de geração dos vizinhos promovidos pelo procedimento "*Vizinhança*" escrito na linha 4 do Algoritmo 17.

3.3.3.2 Geração de vizinhos

A geração dos vizinhos de uma solução é feita como proposto por Ciavotta *et al.* (2013). Um grupo de vizinhos é gerado a partir da escolha aleatória de *k* tarefas da sequência da solução. Cada uma das *k* tarefas, é removida da sequência, e, em seguida, a reinserida em *z* posições adjacentes a esquerda e *z* posições adjacentes a direita, considerando sua posição original. Esta estrutura de vizinhança gera $k \times z \times 2$ vizinhos e é ilustrada através da Figura 28.

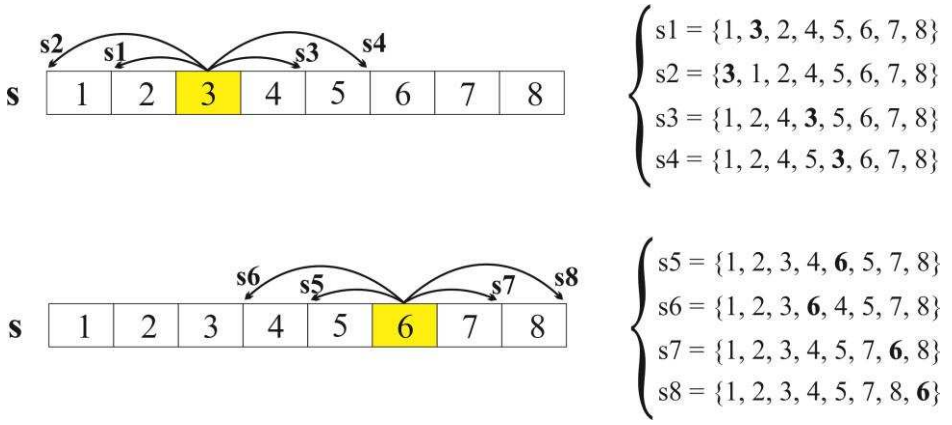

Figura 27: Estrutura de vizinhança (Ciavotta, 2011) - tarefas 3 e 6 são selecionadas e movimentadas para duas posições adjacentes a esquerda e a direita, produzindo os vizinhos *s1*, *s2*, *s3*, *s4*, *s5*, *s6*, *s7* e *s8*

O exemplo apresentado na Figura 28 considera um valor *k*=2 e *z*=2. A sequência base é *s*={1, 2, 3, 4, 5, 6, 7, 8} e as tarefas selecionadas são 3 e 6. A partir do movimento das tarefas

para as posições adjacentes a direita e a esquerda, são gerados 8 vizinhos (*s1*, *s2*, *s3*, *s4*, *s5*, *s6*, *s7* e *s8*).

Ciavotta *et al.* (2013) afirma que a aplicação de uma busca local em um contexto multi-objetivo não é trivial como em um contexto mono-objetivo. Em um contexto multi-objetivo, a avaliação a respeito da melhoria de uma solução é realizada por um critério de dominância, enquanto que em um contexto mono-objetivo, basta simplesmente efetuar a comparação do valor da função objetivo. Porém, a avaliação da dominância é um processo demorado. Por isso, a busca local deve ser rápida e simples. Em alguns casos, onde é gerado uma alta gama de vizinhos e muitas soluções são exploradas, um alto consumo de CPU é imposto devido ao teste de dominância, o que pode fazer com que o algoritmo fique com pouca eficiência na convergência para a fronteira Pareto-ótima.

3.4 Experimentos computacionais

O objetivo desta seção é apresentar os resultados computacionais referentes ao desempenho dos algoritmos heurísticos multi-objetivos propostos para o problema 3sAFS com tempos de preparação dependentes da sequência das tarefas. Com o intuito de extrair a melhor configuração de cada algoritmo, primeiramente foram realizados experimentos de calibração dos algoritmos e, em seguida, foi realizada uma comparação entre algoritmos, utilizando diversas métricas de avaliação. É utilizada uma análise estatística para apoiar a avaliação e a validação dos resultados.

A seguir, são apresentados os critérios utilizados para realização dos experimentos, como a forma de geração das instâncias dos problemas de testes, o critério de parada estabelecido para os algoritmos e as métricas de avaliação de desempenho.

Todos os algoritmos foram codificados em C++ e executados no mesmo ambiente computacional: computador com processador Intel® Core™ i5-3570 CPU @ 3.40GHz, memória de 16GB e sistema operacional Windows 7 Professional de 64 bits.

3.4.1 Geração de instâncias do problema

As instâncias do problema foram geradas aleatoriamente de acordo com Liefooghe *et al.* (2012). Foram considerados problemas de teste com n tarefas ($n \in \{30, 50, 80, 100, 200\}$)

e m máquinas no primeiro estágio ($m \in \{5, 10, 20\}$). Os tempos de processamento são gerados aleatoriamente, de acordo com uma distribuição uniforme: t_{ij}, tt_j, $ta_j \in U[0, 99]$. Os tempos de preparação são uniformemente distribuídos no intervalo [0-49].

As datas de entrega para as tarefas foram geradas aleatoriamente com distribuição uniforme, ao longo do intervalo [$3 \times p$, $(n+2) \times p$], onde p é o valor médio do tempo de processamento anteriormente gerado. Foi utilizado o fator 3, devido ao problema 3sAFS poder ser considerado um *flowshop* em três máquinas. Assim, a data de entrega d_i situa-se, aproximadamente, entre a data de conclusão média da primeira tarefa programada e o tempo médio de conclusão da última tarefa programada (LIEFOOGHE *ET AL.*, 2012).

Para cada configuração $n \times m$, 10 instâncias foram geradas. Portanto, foram gerados um total de 150 diferentes instâncias do problema.

3.4.2 Critério de parada dos algoritmos e métrica de avaliação

Como critério de parada dos algoritmos foi utilizado o tempo máximo de execução, que é definido pela fórmula: $n \times m \times c$. Onde n corresponde ao número de tarefas, m corresponde ao número de máquinas no primeiro estágio e c é uma constante de tempo (em milissegundos). Para os experimentos de calibração foi usado c=50, já para o experimento de comparação dos algoritmos foi utilizado c=100.

Para avaliar a qualidade das soluções não-dominadas alcançados pelos três algoritmos: NSGA-II, NSGA2_IG e NSGA2_IG*, são usadas três medidas multi-objetivos de desempenho: indicador de hipervolume (H*), medida *epsilon* aditiva ($I_{\varepsilon+}$) e Distância média ($d_{média}$).

São denotados por D_1, D_2 e D_3 os conjuntos de soluções não-dominadas (fronteiras de Pareto aproximados) obtidos pelos algoritmos NSGA-II, NSGA2-IG e NSGA2-IG*, respectivamente. A qualidade de cada conjunto D_i (i = 1, 2, 3) é mensurada em relação ao conjunto de referência (*Ref*) de soluções não-dominadas. O conjunto *Ref* é constituído por todas as soluções não dominadas obtidas pelos três algoritmos, ou seja, a fronteira Pareto de soluções não dominadas obtidas de ($D_1 \cup D_2 \cup D_3$). O conjunto *Ref* também é conhecido como "fronteira Pareto ótima", e armazena as soluções não dominadas conhecidas.

As medidas de desempenho utilizadas são definidas a seguir:

- **Indicador de hipervolume (*H**):** mensura a área coberta (ou dominada) por um conjunto D_i , definido da seguinte forma**:**

$$H^*(D_i) = 100 \times \frac{H(Ref) - H(D_i)}{H(Ref)}$$

onde,

H(X) é o hipervolume (área) do espaço de solução dominado pelas soluções do conjunto X, isto é, a porção do espaço objetivo que é dominada por X. Foi considerado a percentagem relativa de desvio $H(D_i)$ com relação a $H(Ref)$.

Os menores valores de H*(D_i) correspondem a mais elevada qualidade das soluções de D_i. O indicador de hipervolume foi introduzido por (ZITZLER e THIELE, 1998).

- **Indicador e*psilon* aditivo unário ($I_{\varepsilon+}$):** medida referente a distância mínima necessária para que um conjunto D_i seja deslocado até o conjunto referência (*Ref*). Esta medida é baseada na diferença de distância de cada solução $s' \in D_i$ em relação as soluções $s \in Ref$. Esta métrica foi proposta por Zitzler *et al.* (2003) e é calculada como se segue:

$$I_{\varepsilon+}(D_i) = \max_{s \in Ref} \left\{ \min_{s' \in D_i} \left(\max_{j=1,\ldots,nObj} N\left(f_j(s') \right) - N\left(f_j(s) \right) \right) \right\}$$

onde,

nObj é o número de objetivos considerados, neste caso, *nObj*=2.

e,

$N\left(f_j(s') \right)$ e $N\left(f_j(s) \right)$, correspondem a normalização (N) da solução $s' \in D_i$ para o objetivo j e a normalização $s \in Ref$ para o objetivo j, respectivamente. A normalização (N) é dada pela seguinte equação:

$$N\left(f_j(x)\right) = \frac{f_j(x) - \min f_j}{\max f_j - \min f_j}$$

onde,

$\min f_j$ e $\max f_j$, são, respectivamente, os valores mínimos e máximos para o objetivo j.

Os menores valores de $I_{\varepsilon+}$ representam uma qualidade mais elevada para as soluções.

- **Distância média ($d_{média}$):** mede a proximidade entre as soluções *s'* ∈ D_i e as soluções *s* ∈ *Ref*. Ele também mede a difusão das soluções no conjunto D_i (KNOWLES e CORNE, 2002). Esta métrica é definida conforme a equação abaixo:

$$d_{média}(D_i) = 100 \times \frac{1}{|Ref|} \sum_{s \in Ref} \min_{s' \in D_i} d(s, s')$$

onde,

$$d(s, s') = \max\left(\frac{f_1(s) - f_1(s')}{\Delta_1}, \frac{f_2(s) - f_2(s')}{\Delta_2}\right)$$

e,

Δ_i corresponde a diferença entre o maior e o menor valor do objetivo f_i, considerando as soluções do conjunto *Ref*.

Quanto menor o valor para $d_{média}$, maior é a qualidade das soluções de D_i.

3.4.3 Calibração dos algoritmos

Para determinar a melhor configuração dos algoritmos foram realizados experimentos de calibração dos algoritmos. Para isso, cada algoritmo foi rodado sobre 50% das instâncias (75) geradas e com o mesmo tempo de execução: $n \times m \times 50$ milisegundos. Para cada instância, cada algoritmo foi executado 10 vezes, totalizando 750 execuções.

3.4.3.1 Calibração do NSGA-II

Os parâmetros do NSGA-II são a probabilidade de realização do cruzamento (*cz*) e a probabilidade de mutação (*mt*). Para o primeiro foram avaliados os valores *cz*={60%, 70%, 80%, 90% e 100%}, enquanto que para o segundo os valores foram *mt*={20%, 30%, 40%, 50% e 60%}. Para este experimento utilizamos apenas o indicador de hipervolume (*H**) como medida de resultado de cada execução.

Inicalmente, o parâmetro *cz* foi avaliado. Para isso o parâmetro *mt* foi fixado com o valor 0, ou seja, não foi considerada mutação. Após a execução do algoritmo e a coleta dos

dados de saída, o indicador de hipervolume foi calculado para cada rodada em cada instância. Com o intuito de verificar se as diferenças são significativas, foi realizada uma análise estatística sobre os dados.

Inicialmente, optou-se por realizar uma análise de variância (ANOVA) parâmetrica (MONTGOMERY, 1997). Porém para que esta análise seja válida, três pressuposições devem ser observadas: homocedasticidade, normalidade e independência. Sendo assim, aplicou-se o teste de *Levene's* para avaliação da homocedasticidade. O resultado "*P-value*" do teste foi 0.00969011. Como o valor é inferior a 0.05, significa que há uma diferença significativa entre os desvios padrões, ou seja, a premissa de homocedasticidade não é válida, inviabilizando o uso da ANOVA.

Uma vez que a ANOVA não é aplicável, utiliza-se o teste não paramétrico de *Kruskal-Wallis*. Este teste compara as medianas dos objetos analisados para determinar se existe uma diferença significativa, com nível de confiança de 95%. O resultado "*P-Value*" do teste *Kruskal-Wallis* foi 0 (zero), o que indica que há diferenças estatísticas significativas, pois trata-se de um valor inferior a 0.05. Porém o teste de *Kruskal Wallis* não especifica quais dos objetos analisados são diferentes. Então, aplica-se um teste de comparações múltiplas não-paramétrico, que compara cada par de média, com um nível de confiança de 95%.

A Tabela 28 mostra o resultado do teste de múltiplas comparações considerando a média do indicador de hipervolume. A primeira coluna mostra os pares de algoritmos que são comparados. A coluna "diferença" mostra a média da amostra do primeiro algoritmo menos a do segundo. A coluna "Limites +/-" mostra um intervalo de incerteza para a diferença. Qualquer par de algoritmos para a qual o valor absoluto da diferença excede o limite é estatisticamente significativo no nível de confiança escolhido de 95% e é indicado por um asterisco (*) na coluna "significativo".

Tabela 28: Resultado do teste de múltiplas comparações entre os diferentes valores de probabilidade de cruzamento do algoritmo NSGA-II

Comparação (Prob. Cruzamento)	***Significante***	***Diferença***	***+/- Limite***
60% - 70%	*	**9,38707**	2,96553
60% - 80%	*	**12,7869**	2,96553
60% - 90%	*	**15,2936**	2,96553
60% - 100%	*	**16,6991**	2,96553
70% - 80%	*	**3,39987**	2,96553
70% - 90%	*	**5,90653**	2,96553
70% - 100%	*	**7,312**	2,96553
80% - 90%		2,50667	2,96553
80% - 100%	*	**3,91213**	2,96553
90% - 100%		1,40547	2,96553

Percebe-se que existem diferenças estatísticas entre os valores *cz*=100% para os demais valores, exceto para *cz*=90%. Isso indica que a variação da probabilidade de cruzamento entre 90% e 100% não resulta em melhorias estatisticamente significantes, porém espera-se definir o parâmetro que, em média, gera os melhores resultados. Para isso, é apresentado um gráfico (Figura 28) de médias e intervalos de *Tukey* da Diferença Honestamente Significativa (HSD), com nível de confiança de 95%. Através dele é possível visualizar qual probabilidade de cruzamento obteve o melhor resultado médio.

O gráfico apresentado na Figura 28 também permite a mesma análise estatística aplicada pelo teste de múltiplas comparações. Uma vez que o intervalo dos objetos analisados não se sobrepõe a qualquer outro intervalo, significa que o objeto em questão é significativamente diferente em relação aos outros. Como pode ser visto, a probabilidade de cruzamento *cz*=100% é interceptada apenas pelo objeto *cz*=90%. Apesar de não possuir diferenças estatistícas significativas entre eles, *cz*=100% corresponde ao ajuste que, em média, gera os melhores resultados.

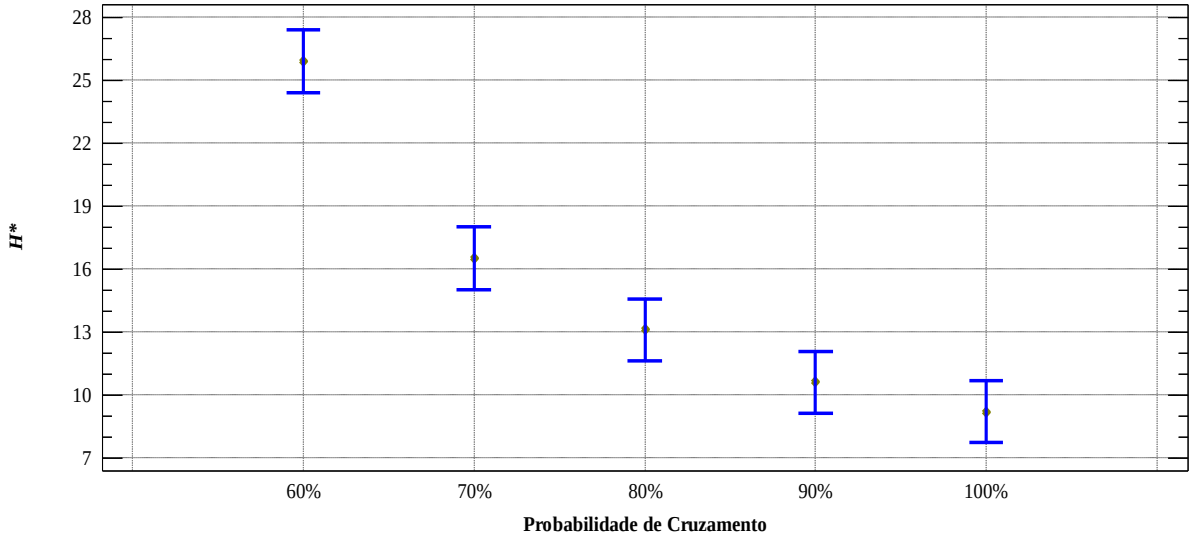

Figura 28: Gráfico de médias e intervalos HSD de *Tukey* com nível de confiança de 95% para as configurações da probabilidade de cruzamento (cz) do algoritmo NSGA-II

A mesma avaliação foi realizada para o parâmetro de *mt*, correspondente a probabilidade de mutação de uma solução. O parâmetro *cz* foi fixado em 100%, enquanto o *mt* variou de acordo com os valores estabelecidos.

A utilização da ANOVA provou-se ser inviável também para este caso, pois o resultado do teste de *Levene's* ficou inferior a 0.05 (*P-value*=0.0469778), indicando que a pressuposição

de homocedasticidade não é satisfeita. Assim, optou-se novamente pela utilização do teste não paramétrico *Kruskal Wallis*, que indicou haver diferença significativa entre, pelo menos, um dos objetos analisados, pelo fato do valor "*P-value*" resultante ser inferior a 0.05 (*P-value*=0,00762945).

O teste de múltiplas comparações é apresentado na Tabela 29, enquanto a Figura 29 mostra o gráfico de médias e intervalo resultante do teste *Tukey* da Diferença Honestamente Significativa (HSD), com nível de confiança de 95%. Tanto pela observação da Tabela 29 quanto pela observação do gráfico da Figura 29, percebe se que o valor *mt*=30% obtém resultados estatisticamente melhores em relação aos valores *mt*=50% e *mt*=60%. Porém, não há diferenças estatísticas significantes para os outros valores *mt*=20% e *mt=4*0%. Contudo, opta-se por utilizar o valor que fornece os melhores resultados médios, ou seja, *mt*=30%.

Tabela 29: Resultado do teste de múltiplas comparações entre os diferentes valores de probabilidade de mutação do algoritmo NSGA-II

Comparação (Prob. Mutação)	***Significante***	***Diferença***	***+/- Limite***
20% - 30%		0,745867	2,08712
20% - 40%		0,4572	2,08712
20% - 50%		-1,42347	2,08712
20% - 60%		-1,45253	2,08712
30% - 40%		-0,288667	2,08712
30% - 50%	*	**-2,16933**	2,08712
30% - 60%	*	**-2,1984**	2,08712
40% - 50%		-1,88067	2,08712
40% - 60%		-1,90973	2,08712
50% - 60%		-0,0290667	2,08712

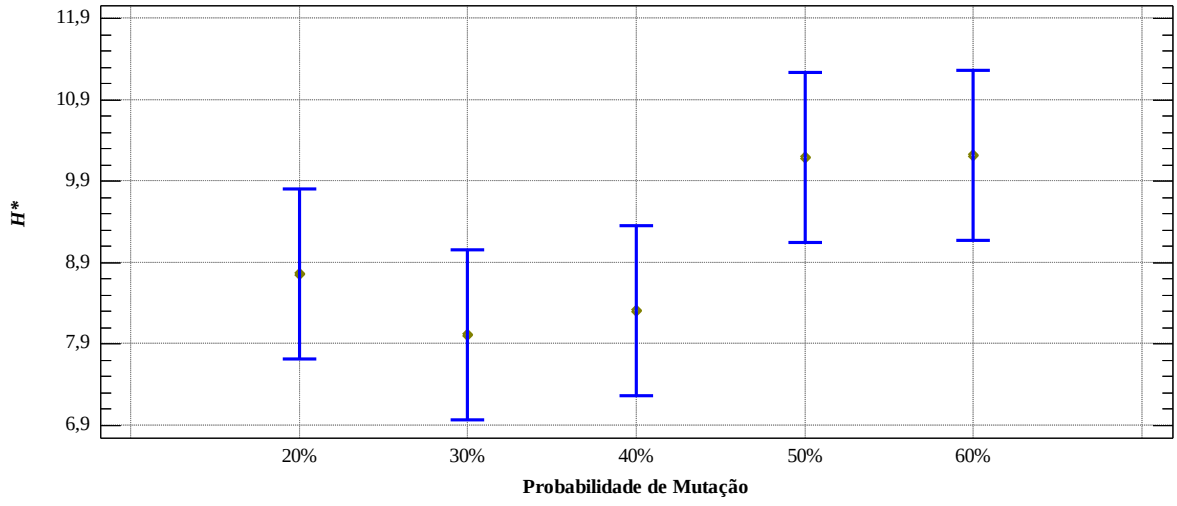

Figura 29: Gráfico de médias e intervalos HSD de *Tukey* com nível de confiança de 95% para as configurações da probabilidade de mutação (*mt*) do algoritmo NSGA-II

Portanto, após a avaliação da análise estatística, o NSGA-II foi configurado com probabilidade de cruzamento igual a 100% e probabilidade de mutação igual a 30%.

3.4.3.2 Calibração do NSGA2_IG

Os parâmetros específicos do NSGA2_IG são os que estão relacionados ao processo de intensificação gulosa promovido pelo IG. Este processo utiliza apenas um parâmetro, que é o parâmetro *d* utilizado na etapa de destruição-construção da sequência de uma solução.

Para este parâmetro utilizou-se 2 diferentes valores d={2, 4}. Os parâmetros relacionados ao processamento do NSGA-II foram regulados com os mesmos valores encontrados na seção anterior (cz=100% e mt=30%).

Após a execução do algoritmo e a coleta dos dados de saída, o indicador de hipervolume foi calculado para cada rodada em cada instância. Com o intuito de verificar se as diferenças são significativas, foi realizada uma análise estatística sobre os dados.

Inicialmente, optou-se por realizar uma análise de variância (ANOVA) parâmetrica (MONTGOMERY, 1997), porém para que esta análise seja válida, três premissas devem ser observadas: homocedasticidade, normalidade e independência. Sendo assim, aplicou-se o teste de *Levene's* para avaliação da homocedasticidade. O resultado "*P-value*" do teste foi 0,0000209749, como o valor é inferior a 0.05, significa que há uma diferença significativa entre os desvios padrões, ou seja, a premissa de homocedasticidade não foi atendida, inviabilizando o uso da ANOVA.

Uma vez que a ANOVA não é aplicável, utiliza-se o teste não paramétrico de *Kruskal-Wallis*. Este teste compara as medianas dos objetos analisados para determinar se existe uma diferença significativa. O resultado "*P-Value*" do teste *Kruskal-Wallis* foi 0.00241728, o que indica que há diferenças estatísticas significativas, pois trata-se de um valor inferior a 0.05. Porém o teste de *Kruskal Wallis* não especifica quais dos objetos analisados são diferentes. Então, aplica-se um teste de comparações múltiplas não-paramétrico, que compara cada par de média, com um nível de confiança de 95%.

O teste indicou que há diferenças estatísticas significantes entre os dois objetos, conforme pode-se ver na Tabela 30. A coluna "diferença" mostra a média da amostra do primeiro algoritmo menos a do segundo, enquanto a coluna "Limites +/-" mostra um intervalo de incerteza para a diferença. Como o valor absoluto da diferença excede o limite, significa que há uma diferença significativa entre os objetos analisados, por isso um asterisco (*) é marcado na coluna "significativo".

Tabela 30: Resultado do teste de múltiplas comparações da calibração de parâmetros do algoritmo NSG2_IG

Comparação	*Significante*	*Diferença*	*+/- Limite*
d=2 – d=4	*	**5,00467**	2,64234

O mesmo resultado também pode ser visto no gráfico (Figura 30) de médias e intervalos de *Tukey* da Diferença Honestamente Significativa (HSD), com nível de confiança de 95%. Uma vez que o intervalo dos objetos analisados não se sobrepõem a qualquer outro intervalo, significa que o objeto em questão é significativamente diferente em relação aos outros. Dessa forma, fica claro que o valor *d*=4 obtém resultados, em uma média geral, melhores. Porém, percebeu-se que o valor *d*=2 obtém resultados 85% melhores quando analisados somente as instâncias de 200 tarefas. O gráfico de intervalos e médias apresentado na Figura 31, mostra este resultado.

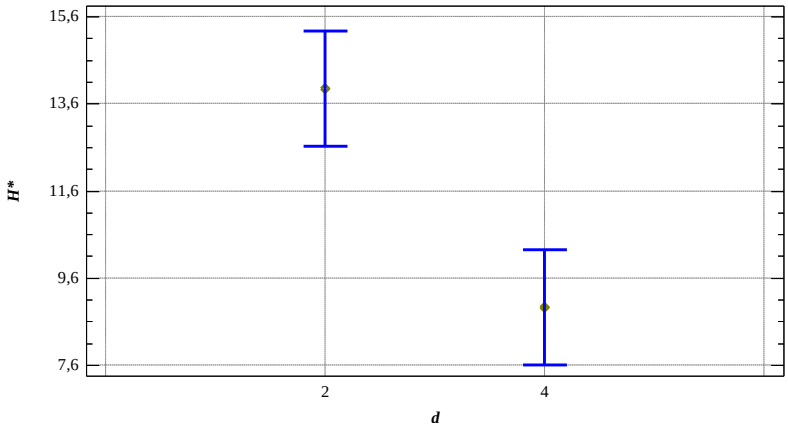

Figura 30: Gráfico de médias e intervalos HSD de *Tukey* com nível de confiança de 95% para a calibração do parâmetro *d* do algoritmo NSGA2_IG

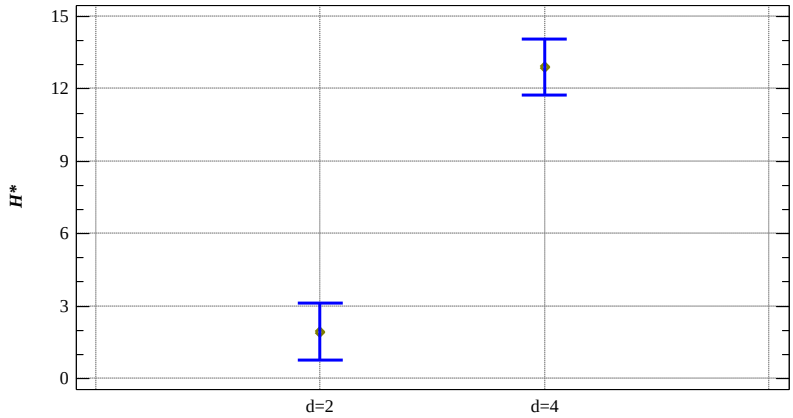

Figura 31: Gráfico de médias e intervalos HSD de *Tukey* com nível de confiança de 95% para a calibração do parâmetro *d* do algoritmo NSGA2_IG para somente instâncias de 200 tarefas

Portanto, o algoritmo NSGA2_IG é configurado com o valor *d*=4, para instâncias menores que 200 tarefas, já o valor *d*=2 é aplicado quando se trata de instâncias maiores ou iguais a 200 tarefas.

3.4.3.3 Calibração do NSGA2_IG*

Para calibração do algoritmo NSGA2_IG* os seguintes parâmetros foram analisados:

- *q*: Determina o percentual de soluções da primeira fronteira F_0 de soluções não dominadas que será submetida ao procedimento IG*;
- *p*: Probabilidade de efetuar a busca local "*BuscaLocalMult*" sobre uma dada solução no procedimento IG*;
- y: Denominador da fração $k = {}^{n}/_{y}$. O fator *k* corresponde ao número de tarefas que serão removidas e reinseridas da sequência de uma solução para geração de vizinhos no procedimento "Vizinhanca" da busca local "*BuscaLocalMult*".

Para definição do melhor conjunto de parâmetros para o algoritmo, foi utilizada neste experimento computacional de calibração, a metodologia de Desenho de Experimentos (DOE), descrita em Montgomery (2006). Nela cada fator é um parâmetro controlado e toda a faixa de valores disponíveis para cada parâmetro são combinadas, gerando então um desenho fatorial completo. Cada combinação de parâmetros é executada e comparada com as outras.

Os seguintes conjuntos de valores foram adotados: $q \in \{40, 60 \text{ e } 80\}$, $p \in \{50, 75, 100\}$ e $y \in \{4, 6, 8\}$. As combinações dos valores dos três parâmetros geram 27 diferentes configurações para o algoritmo. A identificação destas configurações são mostradas na Tabela 31, a seguir.

Tabela 31: Configurações dada pela combinação dos parâmetros *q*, *p* e *y* para o algoritmo NSGA2_IG*

	Parâmetros				**Parâmetros**		
Configuração	***q***	***p***	***y***	**Configuração**	***q***	***p***	***y***
1	0,4	50	4	14	0,6	75	6
2	0,4	50	6	15	0,6	75	8
3	0,4	50	8	16	0,6	100	4
4	0,4	75	4	17	0,6	100	6
5	0,4	75	6	18	0,6	100	8
6	0,4	75	8	19	0,8	50	4
7	0,4	100	4	20	0,8	50	6
8	0,4	100	6	21	0,8	50	8
9	0,4	100	8	22	0,8	75	4
10	0,6	50	4	23	0,8	75	6
11	0,6	50	6	24	0,8	75	8
12	0,6	50	8	25	0,8	100	4
13	0,6	75	4	26	0,8	100	6
				27	0,8	100	8

O parâmetro *d*, relacionado ao processo de destruição-construção do IG*, e o parâmetro *z*, que define a quantidade de posições que uma tarefa será reinserida na sequência de uma solução, durante o processo de geração de vizinhos da busca local, não são calibrados, pois são

definidos aleatoriamente dentro do algoritmo. O parâmetro *d* varia no intervalo [2, 6] e o parâmetro *z* no intervalo de [4, 10].

Para este experimento foi utilizado, como métrica de avaliação, o indicador de hipervolume (*H**) e a indicador *epsilon* aditivo ($I_{\varepsilon+}$). As métricas foram calculadas após cada rodada do algoritmo em cada uma das configurações. Em seguida os resultados foram coletados e avaliados estatisticamente.

Inicialmente, optou-se por realizar uma análise de variância (ANOVA) paramétrica (MONTGOMERY, 1997). Para que a análise através da ANOVA seja válida, três premissas devem ser observadas: homocedasticidade, normalidade e independência. Sendo assim, foi aplicado o teste de *Levene's* sobre os dados relativos ao indicador de hipervolume e também ao indicador *epsilon* aditivo, para avaliação da homocedasticidade, conforme mostrado na Tabela 32. O resultado "*P-value*" para ambas as métricas é superior a 0.05, indicando que não há diferenças significativas entre os desvios padrões, ou seja, a premissa de homocedasticidade é verdadeira.

Tabela 32: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados de calibração do algoritmo NSGA2_IG* de acordo com as métricas de indicador de hipervolume (*H**) e indicador *epsilon* aditivo ($I_{\varepsilon+}$)

	Metric	***Test***	***P-Value***
Levene's	*H**	0,947679	**0,539663**
Levene's	$I_{\varepsilon+}$	1,462	**0,0621271**

A segunda premissa verificada é a normalidade. Para isso é aplicado o teste de *Shapiro-Wilk W*, que pode ser visto na Tabela 33. O resultado do teste demonstra que os dados, em ambas as métricas, não tem uma distribuição de resíduos normal, pois o valor "*P-value*" é inferior a 0.05. Este fato invalida a utilização da ANOVA.

Tabela 33: Teste *Shapiro-Wilk W* para avaliação da pressuposição de normalidade da ANOVA sobre os dados de calibração do algoritmo NSGA2_IG* de acordo com as métricas de indicador de hipervolume (*H**) e indicador *epsilon* aditivo ($I_{\varepsilon+}$)

Test	***Metric***	***Statistic***	***P-Value***
Shapiro-Wilk W	*H**	0,972808	**0,0**
Shapiro-Wilk W	$I_{\varepsilon+}$	0,883164	**0,0**

Devido a impossibilidade de utilização da ANOVA, foi aplicado o teste não parâmetrico de *Kruskal Wallis*, que é uma alternativa não-paramétrica para o caso onde a ANOVA paramétrica não pode ser aplicada. Este teste compara as medianas dos objetos analisados para determinar se existe uma diferença significativa, com nível de confiança de 95%. O valor de interesse é o "*P-value*" resultante, que neste caso foi 0.033312E-8 para o hipervolume e 0.02382E-8 a medida de *epsilon*. Quando o "*P-value*" é inferior a 0.05, indica que há diferença

significativa entre pelo menos dois dos objetos analisados, porém o teste de *Kruskal Wallis* não especifica quais são. Então, aplica-se um teste de múltiplas comparações não-paramétrico, que compara cada par de média, com um nível de confiança de 95%.

As Tabelas 34 e 35 mostram uma visualização alternativa para o resultado do teste de múltiplas comparações, respectivamente, para as métricas de indicador de hipervolume e indicador *epsilon* aditivo, pois a visualização da comparação de pares para as 27 configurações produz uma tabela de 352 linhas, o que torna difícil a avaliação. A Tabela 34, mostra o resultado da média do indicador de hipervolume alcançado por cada configuração na coluna "H* médio". Também é possível visualizar os grupos de configurações que possuem médias homogêneas, ou seja, aquelas que não possuem diferenças estatísticas significantes entre si. Isso pode ser visto através da coluna "Grupos homogêneos": as configurações que possuem o "X" na mesma direção vertical pertencem ao mesmo grupo. Além disso, o número de amostras utilizadas no teste é mostrado na coluna "amostra". A mesma avaliação é feita na Tabela 35, porém ela corresponde aos dados referente ao indicador *epsilon* aditivo.

Tabela 34: Resultado do teste de múltiplas comparações, em uma apresentação alternativa, sobre os dados de calibração do algoritmo NSGA2_IG*, considerando a métrica de indicador de hipervolume (*H**)

Configuração	*Amostra*	*H* médio*	*Grupos Homogêneos*
25	75	11,0291	X
27	75	11,4289	XX
26	75	11,6804	XXX
22	75	11,9429	XXXX
13	75	11,9587	XXXX
23	75	12,09	XXXXX
17	75	12,0969	XXXXX
21	75	12,3719	XXXXXX
16	75	12,3987	XXXXXX
20	75	12,4808	XXXXX
19	75	12,5151	XXXXX
14	75	12,6861	XXXXXX
10	75	12,6961	XXXXXX
7	75	12,9541	XXXXXX
18	75	12,9649	XXXXXX
12	75	12,99	XXXXXX
15	75	13,0119	XXXXXX
24	75	13,1781	XXXXX
8	75	13,2327	XXXXX
4	75	13,326	XXXXXX
5	75	13,4972	XXXXXX
1	75	13,6612	XXXXX
11	75	14,0459	XXXX
6	75	14,1216	XXXX
9	75	14,3691	XXX
3	75	14,7064	XX
2	75	14,8876	X

Tabela 35: Resultado do teste de múltiplas comparações, em uma apresentação alternativa, sobre os dados de calibração do algoritmo NSGA2_IG*, considerando a métrica indicador *epsilon* aditivo ($I_{\varepsilon+}$)

Configuração	*Amostra*	*em médio*	*Grupos Homogêneos*
25	75	15,1199	X
27	75	16,4575	XX
26	75	16,7012	XXX
22	75	16,8234	XXX
23	75	17,0772	XXX
20	75	17,4304	XXXX
17	75	17,6045	XXXX
24	75	17,7326	XXXX
16	75	17,8726	XXXXX
18	75	17,9598	XXXXX
21	75	18,0555	XXXXX
15	75	18,1567	XXXXXX
13	75	18,7365	XXXXX
19	75	18,9671	XXXXXX
14	75	18,9681	XXXXXX
12	75	19,0697	XXXXXX
10	75	19,5684	XXXXXXX
7	75	20,1006	XXXXXX
8	75	20,8117	XXXXXX
11	75	20,9083	XXXXXX
5	75	21,3437	XXXXX
1	75	21,549	XXXX
4	75	21,5645	XXXX
6	75	21,587	XXXX
9	75	22,3425	XXX
3	75	22,6783	XX
2	75	24,0325	X

De acordo com a métrica de hipervolume, as melhores configurações são, respectivamente: 25, 27, 26, 22, 13, 23, 17, 21 e 16. Entre estas configurações não existem diferenças estatísticas significantes, sendo a configuração 25 a que obteve o menor indicador de hipervolume, em média. Quando analisado o indicador *epsilon* aditivo, percebe-se que as melhores configurações são, respectivamente: 25, 27, 26, 22, 23, 20, 17, 24, 16, 18, 21 e 15. Assim como na métrica anterior, a configuração 25 é a que obteve a menor média.

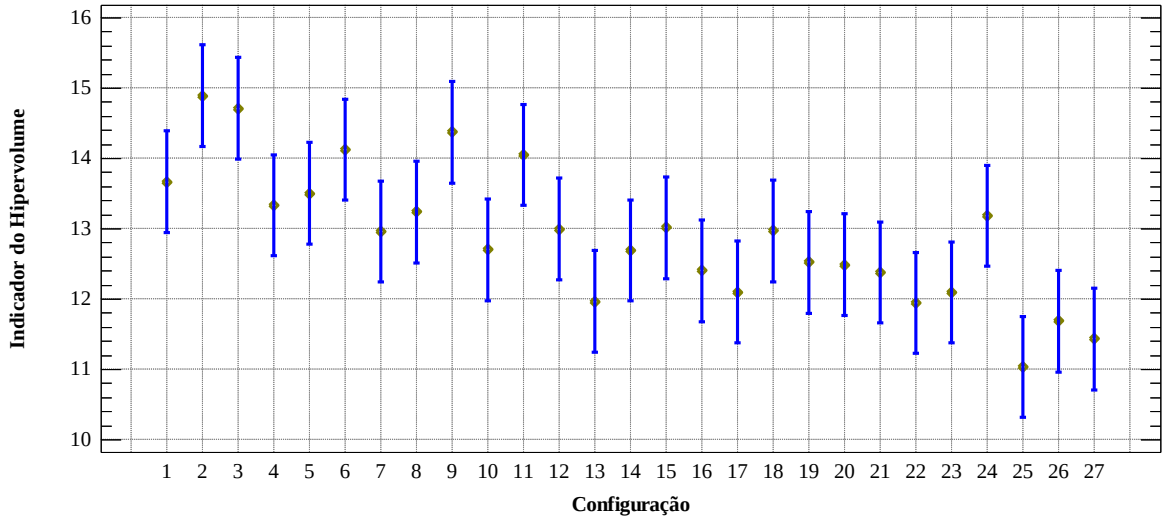

Figura 32: Gráfico de médias e intervalos HSD de *Tukey* com nível de confiança de 95% para as configurações do algoritmo NSGA2_IG* segundo a métrica indicador de hipervolume (*H**)

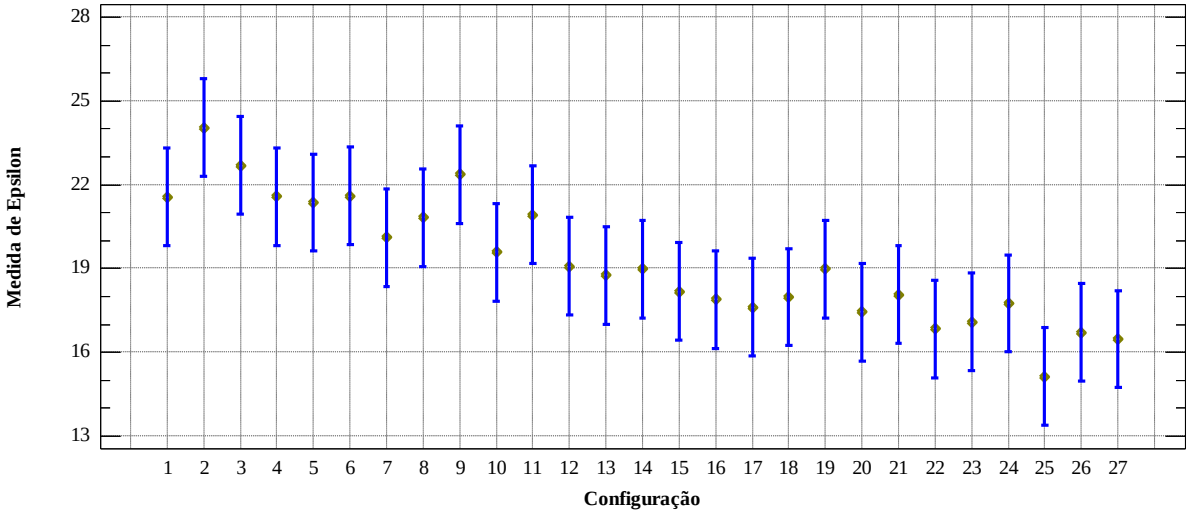

Figura 33: Gráfico de médias e intervalos HSD de *Tukey* com nível de confiança de 95% para as configurações do algoritmo NSGA2_IG* segundo a métrica indicador *epsilon* aditivo ($I_{\varepsilon+}$)

Portanto, a melhor configuração, com base em ambas as métricas, é a configuração 25. Apesar de outras configurações em mesmo nível estatístico, ela foi a que obteve os melhores resultados médios. Este mesmo resultado pode ser visto através do gráfico de médias e intervalos resultante do teste de *Tukey* da Diferença Honestamente Significativa (HSD), com

nível de confiança de 95%, apresentado na Figura 32 e 33, respectivamente, para as métricas de indicador de hipervolume e indicador *epsilon* aditivo. Nota-se que o intervalo da configuração 25 é menor para ambas as métricas. Assim, foi escolhida a configuração 25 que corresponde aos valores: q=0.8, p=100 e y=4.

3.4.4 Comparação dos algoritmos propostos

Nesta seção é realizado um experimento de comparação dos algoritmos NSAG-II, NSGA2_IG e NSGA2_IG* em relação ao desempenho. Para isso, cada algoritmo foi regulado conforme sua melhor configuração, executado em ambientes computacionais iguais, sobre todo o conjunto de instâncias produzidas e com mesmo tempo de CPU ($n \times m \times 100$ milisegundos). Cada algoritmo, em cada instância, foi rodado 10 vezes, totalizando 1500 execuções de cada uma.

Para este experimento foram utilizadas as métricas de indicador de hipervolume (H^*), indicador *epsilon* aditivo ($I_{\varepsilon+}$) e distância média ($d_{\text{média}}$). As métricas foram calculadas após cada rodada do algoritmo em cada uma das configurações. Em seguida, os resultados foram coletados e avaliados estatisticamente.

A Tabela 36 mostra os resultados médios dos algoritmos para cada uma das métricas.

Tabela 36: Resultados médios dos algoritmos NSGA-II, NSGA2_IG e NSGA2_IG* nas métricas de hipervolume (H^*), indicador *epsilon* aditivo ($I_{\varepsilon+}$) e distância média (D_{media}) para todos os conjuntos de instâncias

Instância		**NSGA-II**			**NSGA2_IG**			**NSGA2_IG***		
n	***m***	***H****	$I_{\varepsilon+}$	D_{media}	***H****	$I_{\varepsilon+}$	D_{media}	***H****	$I_{\varepsilon+}$	D_{media}
30	**5**	42,29	59,036	25,959	10,81	25,752	5,815	0,57	4,726	0,376
30	**10**	38,54	51,775	25,581	11,72	22,921	7,572	0,25	1,503	0,140
30	**20**	40,40	68,821	30,646	10,28	24,690	7,605	1,04	4,567	0,339
50	**5**	61,78	85,554	41,354	16,35	33,198	8,765	0,38	2,965	0,282
50	**10**	52,71	71,818	33,935	14,33	30,337	8,402	0,36	1,568	0,136
50	**20**	52,29	89,312	40,934	15,57	37,817	11,340	0,52	1,798	0,068
80	**5**	62,33	89,799	41,369	23,00	43,121	13,004	0,00	0,170	0,002
80	**10**	61,69	108,272	42,400	18,43	43,802	9,936	0,06	0,419	0,010
80	**20**	58,00	109,937	48,847	13,87	34,748	9,245	0,20	1,267	0,111
100	**5**	64,04	80,138	40,580	20,54	38,468	10,999	0,00	0,000	0,000
100	**10**	59,30	84,550	39,630	19,00	40,869	10,523	0,04	0,591	0,067
100	**20**	59,78	257,213	51,023	18,33	82,348	12,885	0,17	1,475	0,101
200	**5**	60,94	63,713	36,355	14,84	28,115	7,816	5,72	13,778	2,312
200	**10**	54,45	60,549	32,986	11,05	32,123	7,235	5,81	11,084	1,851
200	**20**	48,02	85,261	33,657	13,28	44,100	8,829	2,52	7,587	0,868
Média		**54,44**	**91,050**	**37,684**	**15,43**	**37,494**	**9,331**	**1,17**	**3,567**	**0,444**

Percebe-se que o algoritmo NSGA2_IG* obteve melhores resultados em todas as métricas e para todos os tamanhos de instância. De acordo com o indicador de hipervolume

(*H**), o NSGA2_IG* obtém, em média, resultados 46,5 e 13,1 vezes melhores que o NSGA-II e o NSGA2_IG, respectivamente. Já em relação ao indicador *epsilon* aditivo ($I_{\varepsilon+}$), a melhoria do NSGA2_IG* é, em média, 25,5 e 10,5 vezes, em relação ao NSGA-II e NSGA2_IG, respectivamente. E, por fim, analisando a métrica de distância média, a melhoria média do NSGA2_IG* é de 84,9 e 21,1 vezes, em relação ao NSGA-II e ao NSGA2_IG, respectivamente.

O resultado obtido indica que a aplicação da busca local em descida, associada ao processo de intensificação gulosa, nas soluções não dominadas da primeira fronteira do NSGA-II, contribui fortemente para aumento da qualidade das soluções encontradas.

As Figuras 34, 35 e 36 mostram a convergência para a fronteira Pareto do conjunto referência (*Ref*). Nota-se que o processo de intensificação gulosa promovida pelo IG (através do NSGA2_IG) produz grande deslocamento da fronteira em relação ao NSGA-II. Já a adição da busca local em descida e o aumento do número de soluções exploradas na primeira fronteira, proposto pelo NSGA2_IG*, aumenta ainda mais este deslocamento, fazendo com que a distância entre a fronteira do conjunto referência em relação a fronteira aproximada do algoritmo fiquem bem próximas.

A Figura 34 mostra uma instância com *n*=30 tarefas e *m*=20 máquinas; já a Figura 35 mostra uma instância de *n*=80 tarefas e *m*=20 máquinas, e, por fim, na Figura 36, tem-se uma instância de *n*=200 tarefas e *m*=20 máquinas.

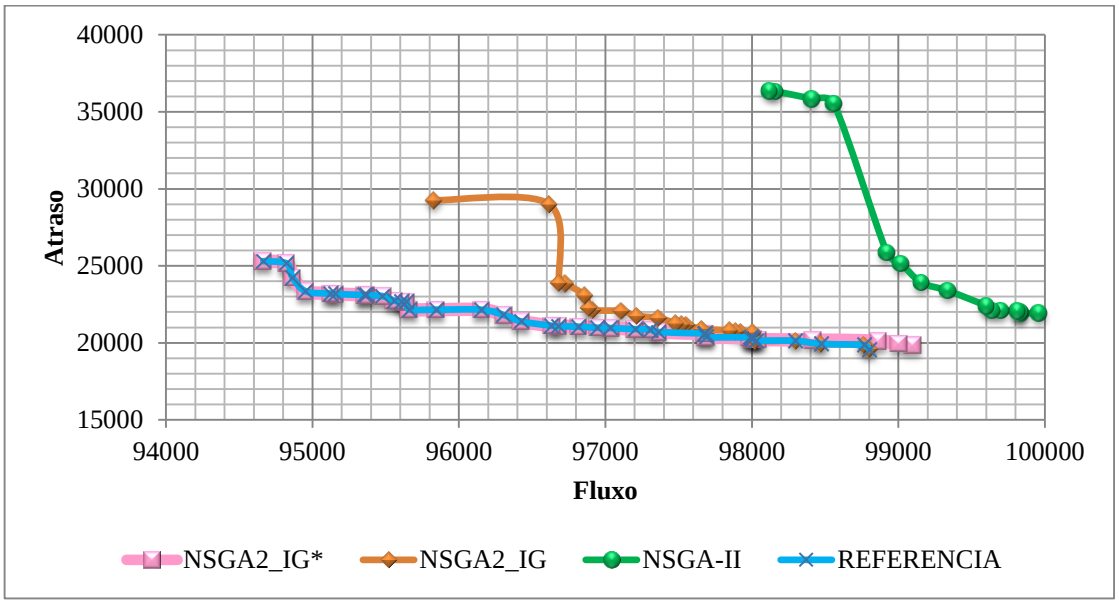

Figura 34: Fronteiras Pareto aproximadas dos algoritmos e do conjunto referência (*Ref*) para uma instância de *n*=30 tarefas e *m*=20 máquinas

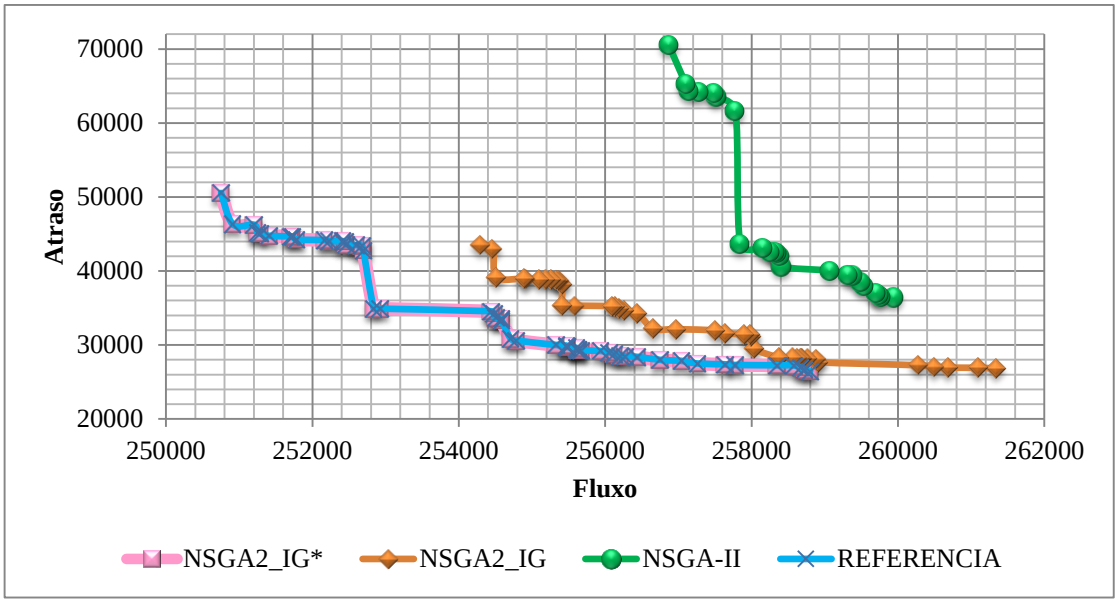

Figura 35: Fronteiras Pareto aproximadas dos algoritmos e do conjunto referência (*Ref*) para uma instância de *n*=80 tarefas e *m*=20 máquinas

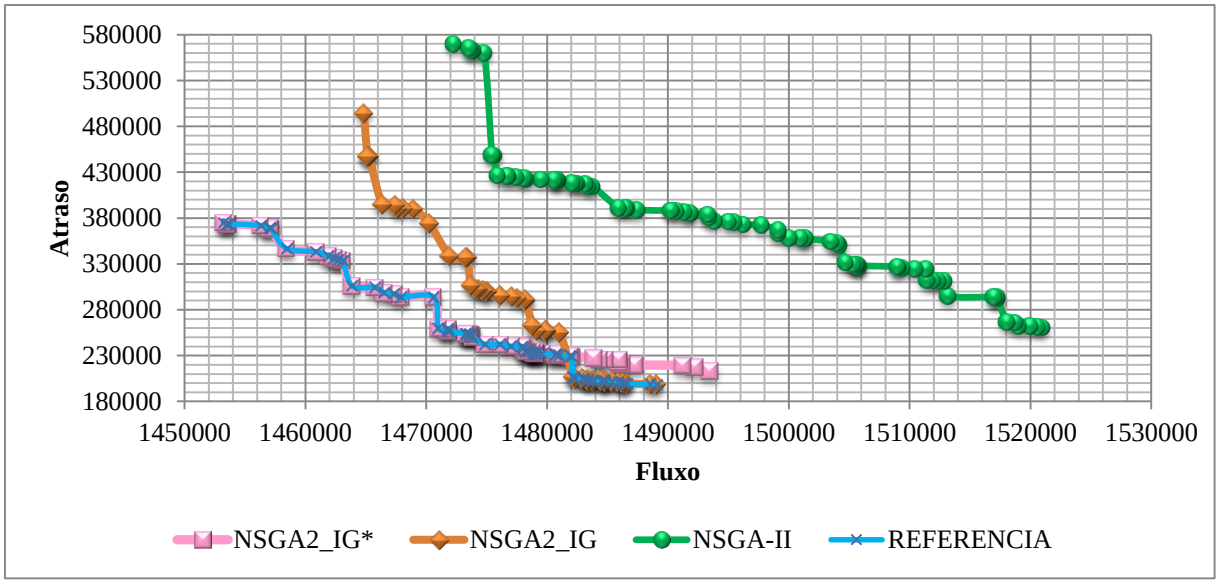

Figura 36: Fronteiras Pareto aproximadas dos algoritmos e do conjunto referência (*Ref*) para uma instância de *n*=200 tarefas e *m*=20 máquinas

Para validar os resultados e verificar se as diferenças entre os algoritmos são significativas, foi realizada um análise estatística, considerando as três métricas usadas. Inicialmente, optou-se por realizar uma análise de variância (ANOVA) parâmetrica (MONTGOMERY, 1997). Para que a análise através da ANOVA seja válida, três pressuposições devem ser observadas: homocedasticidade, normalidade e independência. Sendo assim, foi aplicado o teste de *Levene's* sobre os dados para avaliação da homocedasticidade. O resultado "*P-value*" para todas as métricas foi inferior a 0.05 (Tabela 37), indicando que há diferenças significativas entre os desvios padrões, ou seja, a pressuposição de homocedasticidade é rejeitada, invalidando o uso da ANOVA.

Tabela 37: Teste *Levene's* para avaliação da pressuposição de homocedasticidade da ANOVA sobre os dados dos resultados obtidos pelos algoritmos, para as métricas: indicador de hipervolume (*H**), indicador *epsilon* aditivo ($I_{\varepsilon+}$) e distância média (D_{media})

	Metric	***Test***	***P-Value***
Levene's	*H**	78,3693	**0**
Levene's	*me*	9,01049	**0,000145753**
Levene's	D_{media}	76,4165	**0**

Devido a impossibilidade de utilização da ANOVA, foi aplicado o teste não parâmetrico de *Kruskal Wallis*, que é uma alternativa não-paramétrica para o caso onde a ANOVA paramétrica não pode ser aplicada. Este teste compara as medianas dos objetos para determinar se existe uma diferença significativa, com nível de confiança de 95%. O valor de interesse é o "*P-value*" resultante, que neste caso foi 0 (zero) para todas as métricas. Quando o "*P-value*" é inferior a 0.05, indica que há diferença significativa entre pelo menos dois dos objetos analisados, isto é, todas as métricas apresentam diferenças significantes. Porém, o teste de *Kruskal Wallis* não especifica quais objetos são diferentes. Então, aplica-se um teste de múltiplas comparações não-paramétrico, que compara cada par de média, com um nível de confiança de 95%.

O resultado do teste de múltiplas comparações está disponível na Tabela 38. A primeira coluna apresenta a métrica utilizada na comparação, enquanto a segunda coluna mostra os pares de algoritmos comparados. A coluna "diferença" mostra a média da amostra do primeiro algoritmo menos a do segundo. A coluna "Limites +/" mostra um intervalo de incerteza para a diferença. Qualquer par de algoritmos, para qual o valor absoluto da diferença exceda o limite, é estatisticamente significativo no nível de confiança escolhido de 95% e é indicado por um asterisco (*) na coluna "significativo".

Tabela 38: Resultado do teste de múltiplas comparações sobre os dados de resultados obtidos pelos algoritmos em função das métricas de indicador de hipervolume (*H**), indicador *epsilon* aditivo ($I_{\varepsilon+}$) e distância média (*Dmédia*)

Métrica	**Comparação**	**Significante**	**Diferença**	**+/- Limite**
H*	NSGA-II - NSGA2_IG	*	**39,0103**	1,69417
	NSGA-II - NSGA2_IG*	*	**53,2624**	1,69417
	NSGA2_IG - NSGA2_IG*	*	**14,2521**	1,69417
$I_{\varepsilon+}$	NSGA-II - NSGA2_IG	*	**53,5559**	15,0939
	NSGA-II - NSGA2_IG*	*	**87,4833**	15,0939
	NSGA2_IG - NSGA2_IG*	*	**33,9274**	15,0939
$D_{média}$	NSGA-II - NSGA2_IG	*	**28,3523**	1,63202
	NSGA-II - NSGA2_IG*	*	**37,2396**	1,63202
	NSGA2_IG - NSGA2_IG*	*	**8,88725**	1,63202

Percebe-se que existem diferenças significativas entre todos os algoritmos, para todas as métricas. A visualização a respeito da diferença média entre os resultados obtidos pelos algoritmos pode ser vista através do gráfico de médias e intervalos resultante do teste de *Tukey*

da Diferença Honestamente Significativa (HSD), com nível de confiança de 95%. Além da visualização das médias, este gráfico valida o resultado apresentado pelo teste de múltiplas comparações, pois quando o intervalo de um objeto analisado não sobrepõe aos outros intervalos, conclui-se que há diferença significativamente entre as médias.

Foi gerado um gráfico de médias e intervalos para cada uma das métricas usadas. Eles podem ser vistos nas Figuras 37, 38 e 39, respectivamente, para as métricas, indicador de hipervolume (*H**), indicador *epsilon* aditivo ($I_{\varepsilon+}$) e distância média ($D_{média}$).

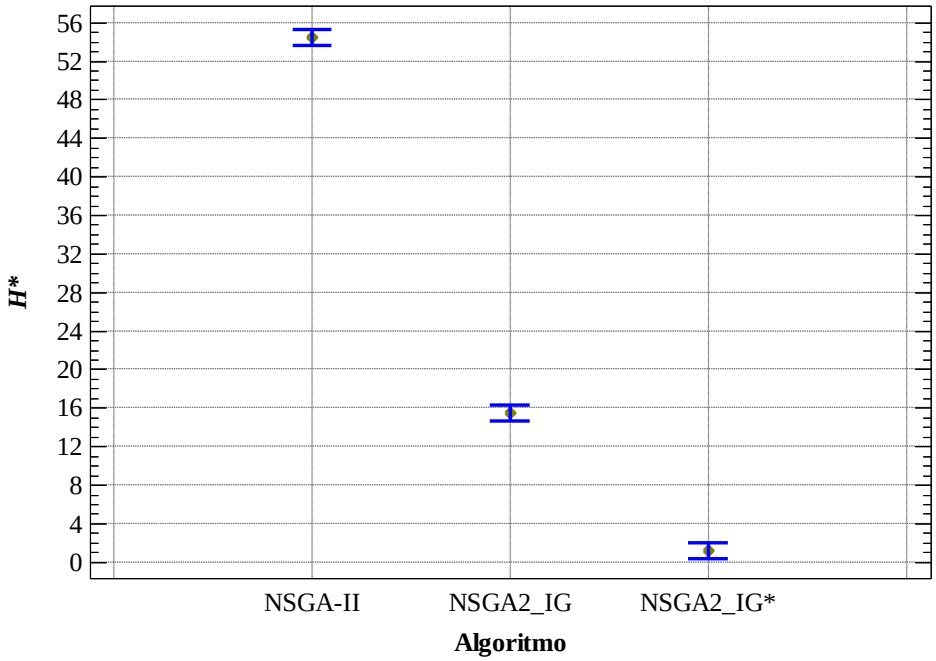

Figura 37: Gráfico de médias e intervalos HSD do teste de *Tukey* com nível de confiança de 95% para os algoritmos NSGA-II, NSGA2_IG e NSGA2_IG* de acordo com a métrica indicador de hipervolume (*H**)

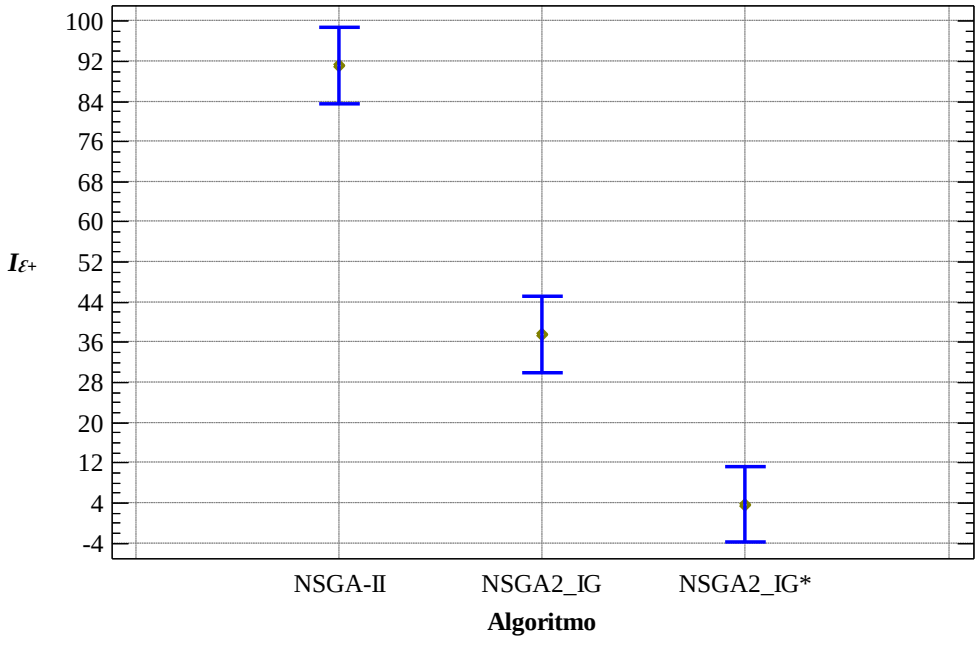

Figura 38: Gráfico de médias e intervalos HSD do teste de *Tukey* com nível de confiança de 95% para os algoritmos NSGA-II, NSGA2_IG e NSGA2_IG* de acordo com a métrica indicador epsilon aditivo ($I_{\varepsilon+}$)

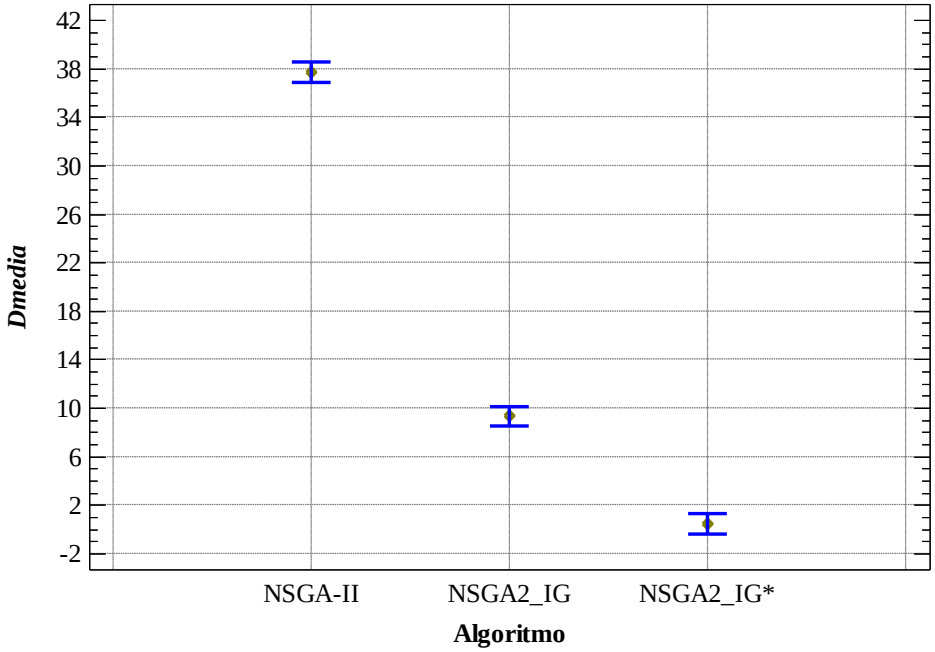

Figura 39: Gráfico de médias e intervalos HSD do teste de *Tukey* com nível de confiança de 95% para os algoritmos NSGA-II, NSGA2_IG e NSGA2_IG* de acordo com a métrica de distância média ($D_{média}$)

Pode-se observar nos gráficos das Figuras 37, 38 e 39 que os intervalos não se cruzam para nenhum dos algoritmos, em todas as métricas, mostrando que as diferenças entre os algoritmos são estatisticamente significativas. Isso possibilita afirmar que os resultados produzidos pelo NSGA2_IG são significativamente melhores que os resultados produzidos pelo NSGA-II. O mesmo ocorre em relação ao NSGA2_IG*, que obtém resultados melhores que o NSGA2_IG.

3.5 Conclusões

Este artigo examina estratégias de resolução multi-objetivo para o problema de programação da produção em um ambiente *assembly flowshop* com três estágios, onde o primeiro estágio é responsável pela fabricação de partes da tarefa, constituído por diversas máquinas paralelas não idênticas; o segundo estágio tem a responsabilidade de transportar as partes produzidas e é composto por uma máquina simples; e, por fim, o terceiro estágio é destinado à montagem final da tarefa, também constituído por uma máquina simples. Para deixar o problema ainda mais realístico, são considerados tempos de preparação das máquinas dependentes da sequência de execução das tarefas em todos os estágios do problema.

A abordagem, aqui realizada minimiza simultaneamente dois objetivos: fluxo total e atraso total das tarefas. É utilizado o conceito de dominância para geração de um conjunto de soluções Pareto (soluções não dominadas) que proporciona ao administrador um conjunto de boas soluções que podem ser escolhidas de acordo com situações específicas.

A literatura mostra que este é um problema complexo de otimização combinatória com ampla aplicação nas indústrias de manufatura. Em virtude de sua natureza NP-Difícil e, também, pela abordagem multi-objetivo, este trabalho focou em estratégias de resolução heurísticas. Foi aplicado um tradicional algoritmo da literatura, NSGA-II, e, também, os algoritmos híbridos propostos, NSGA2_IG e NSGA2_IG*, que combinam os conceitos explorados pelo próprio NSGA-II e pela metaheurística *Iterated Greedy*. Estes algoritmos propõem uma intensificação gulosa na primeira fronteira Pareto (F_0), gerada pelo NSGA-II e, também, a aplicação de uma busca local em descida.

Foram realizados experimentos computacionais para calibração dos algoritmos e também para compará-los. Para análise dos resultados foram utilizadas três conhecidas métricas de avaliação de qualidade: hipervolume, indicador unário *epsilon* aditivo e distâncias média. Além disso, todas as análises de resultados foram apoiadas por experimentos estatísticos.

Os resultados destes experimentos comprovam o sucesso das ideias aplicadas através dos algoritmos NSGA2_IG e NSGA2_IG*. É indicado que a intensificação gulosa proposta pelo NSGA2_IG, gera resultados, em média, até 4 vezes melhores que o NSGA-II (considerando métrica de distância média). A aplicação da busca local em descida, promovida pelo algoritmo NSGA2_IG*, reflete em uma melhora, em relação ao NSGA-II, de 85 vezes (também considerando a métrica de distância média). As análises estatísticas mostram que, para todas as métricas, os algoritmos propostos melhoram, de forma significativa, os resultados alcançados pelo NSGA-II.

Portanto, conclui-se que as ideias aqui abordadas são de grande contribuição para a construção de algoritmos heurísticos multi-objetivos para resolução do 3sAFS. É sugerido que as pesquisas neste campo continuem sendo desenvolvidas. Por isso, a seção seguinte lista um conjunto de sugestões de trabalhos futuros.

3.6 Trabalhos Futuros

Os seguintes trabalhos futuros são indicados:

a) Comparação do algoritmo NSGA2_IG* com outros algoritmos da literatura, como por exemplo, $MOSAII_M$ (VARADHARAJAN e RAJENDRAN, 2005) e o RIPG (MINELLA ET AL., 2011);

b) Testar outras estruturas de vizinhança para buca local em descida aqui proposta;

c) Explorar outras variações do problema, como por exemplo, o aumento do número de máquinas nos estágios de transporte e montagem ou o uso de um *flowshop* híbrido no primeiro estágio.

d) Adicionar aos experimentos computacionais outras métricas de avaliação de qualidade.

3.7 Referências bibliográficas

Al-Anzi, F. S., & Allahverdi, A. (2006). A hybrid tabu search heuristic for the two-stage assembly scheduling problem. International Journal of Operations Research, 3(2), 109-119.

Al-Anzi, F. S., & Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. European Journal of Operational Research, 182(1), 80-94.

Allahverdi, A., & Al-Anzi, F. S. (2006). Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times. International Journal of Production Research, 44(22), 4713-4735.

Allahverdi, A., & Al-Anzi, F. S. (2008). The two-stage assembly flowshop scheduling problem with bicriteria of makespan and mean completion time.The International Journal of Advanced Manufacturing Technology, 37(1-2), 166-177

Allahverdi, A., Ng, C. T., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 187(3), 985-1032.

Allahverdi, A., & Aydilek, H. (2014). The two stage assembly flowshop scheduling problem to minimize total tardiness. Journal of Intelligent Manufacturing, 1-13.

Andrés, C., & Hatami, S. (2011, September). The three stage assembly permutation flowshop scheduling problem. In V international conference on industrial engineering and industrial management (pp. 867-875).

Arroyo, J. E. C., & Armentano, V. A. (2004). A partial enumeration heuristic for multi-objective flowshop scheduling problems. Journal of the Operational Research Society, 55(9), 1000-1007.

Arroyo, J. E. C., dos Santos Ottoni, R., & de Paiva Oliveira, A. (2011). Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows. Electronic Notes in Theoretical Computer Science, 281, 5-19.

Campos, S. C., & Arroyo, J. E. C. (2014, July). NSGA-II with iterated greedy for a bi-objective three-stage assembly flowshop scheduling problem. In Proceedings of the 2014 conference on Genetic and evolutionary computation (pp. 429-436). ACM.

Campos, S. C., Arroyo, J. E. C., & Gonçalves, L. B. (2013). Uma heuristica grasp-vnd para o problema de sequenciamento de tarefas num ambiente assembly flowshop com três estágios e tempos de setup dependentes da sequência. In Proceedings of the XLV Brazilian Symposium of Operational Research.(Natal-RN, Brazil, Setember 16-19, 2013).

Ciavotta, M., Minella, G., & Ruiz, R. (2013). Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study. European Journal of Operational Research, 227(2), 301-313.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. Evolutionary Computation, IEEE Transactions on, 6(2), 182-197.

Deb, K., Steuer, R. E., Tewari, R., & Tewari, R. (2011, January). Bi-objective portfolio optimization using a customized hybrid NSGA-II procedure. In Evolutionary Multi-criterion Optimization (pp. 358-373). Springer Berlin Heidelberg..

Fattahi, P., Hosseini, S. M. H., & Jolai, F. (2013). A mathematical model and extension algorithm for assembly flexible flow shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 65(5-8), 787-802.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, 1(2), 117-129.

Guo, Y., Chen, Z. R., Ruan, Y. L., & Zhang, J. (2012, October). Application of NSGA-II with local search to multi-dock cross-docking sheduling problem. In Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on (pp. 779-784). IEEE.

Hariri, A. M. A., & Potts, C. N. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. European Journal of Operational Research, 103(3), 547-556.

Hatami, S., Ebrahimnejad, S., Tavakkoli-Moghaddam, R., & Maboudian, Y. (2010). Two meta-heuristics for three-stage assembly flowshop scheduling with sequence-dependent setup times. The International Journal of Advanced Manufacturing Technology, 50(9-12), 1153-1164.

Ishibuchi, H., Yoshida, T., & Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. Evolutionary Computation, IEEE Transactions on, 7(2), 204-223.

Knowles, J., & Corne, D. (2002, May). On metrics for comparing nondominated sets. In Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on (Vol. 1, pp. 711-716). IEEE.

Koulamas, C., & J Kyparisis, G. (2001). The three-stage assembly flowshop scheduling problem. Computers & Operations Research, 28(7), 689-704.

Liefooghe, A., Basseur, M., Humeau, J., Jourdan, L., & Talbi, E. G. (2012). On optimizing a bi-objective flowshop scheduling problem in an uncertain environment. Computers & Mathematics with Applications, 64(12), 3747-3762.

Maleki-Darounkolaei, A., Modiri, M., Tavakkoli-Moghaddam, R., & Seyyedi, I. (2012). A three-stage assembly flow shop scheduling problem with blocking and sequence-dependent set up times. Journal of Industrial Engineering International, 8(1), 1-7.

Montgomery, D. C. and Montgomery, D. C. (1997). Design and analysis of experiments (Vol. 7). New York: Wiley.

Mozdgir, A., Fatemi Ghomi, S. M. T., Jolai, F., & Navaei, J. (2013). Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times. International Journal of Production Research, 51(12), 3625-3642.

Minella, G., Ruiz, R., & Ciavotta, M. (2011). Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems.Computers & Operations Research, 38(11), 1521-1533.

Navaei, J., Ghomi, S. F., Jolai, F., Shiraqai, M. E., & Hidaji, H. (2013). Two-stage flow-shop scheduling problem with non-identical second stage assembly machines. The International Journal of Advanced Manufacturing Technology, 69(9-12), 2215-2226.

Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11, 1 (1983), 91-95.

Pan, Q. K., & Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. Omega, 44, 41-50.

Pires, D. F., Antunes, C. H., & Martins, A. G. (2012). NSGA-II with local search for a multi-objective reactive power compensation problem. International Journal of Electrical Power & Energy Systems, 43(1), 313-324.

Potts, C. N., Sevast'Janov, S. V., Strusevich, V. A., Van Wassenhove, L. N., & Zwaneveld, C. M. (1995). The two-stage assembly scheduling problem: complexity and approximation. Operations Research, 43(2), 346-355.

Ruiz, R., Maroto, C., & Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. European Journal of Operational Research, 165(1), 34-54.

Ruiz, R., & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research, 187(3), 1143-1159.

Seidgar, H., Kiani, M., Abedi, M., & Fazlollahtabar, H. (2014). An efficient imperialist competitive algorithm for scheduling in the two-stage assembly flow shop problem. International Journal of Production Research, 52(4), 1240-1256.

Shokrollahpour, E., Zandieh, M., & Dorri, B. (2011). A novel imperialist competitive algorithm for bi-criteria scheduling of the assembly flowshop problem. International Journal of Production Research, 49(11), 3087-3103.

Sivanandam, S. N., & Deepa, S. N. (2008). Genetic Algorithm Optimization Problems (pp. 165-209). Springer Berlin Heidelberg.

Srinivas, N., & Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary computation, 2(3), 221-248.

Tajbakhsh, Z., Fattahi, P., & Behnamian, J. (2013). Multi-objective assembly permutation flow shop scheduling problem: a mathematical model and a meta-heuristic algorithm. Journal of the Operational Research Society..

Torabzadeh, E., & Zandieh, M. (2010). Cloud theory-based simulated annealing approach for scheduling in the two-stage assembly flowshop.Advances in Engineering Software, 41(10), 1238-1243.

Varadharajan, T. K., & Rajendran, C. (2005). A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research, 167(3), 772-795.

Yan, H. S., Wan, X. Q., & Xiong, F. L. (2014). A hybrid electromagnetism-like algorithm for two-stage assembly flow shop scheduling problem.International Journal of Production Research, (ahead-of-print), 1-14.

Zitzler, E., & Thiele, L. (1998, January). Multiobjective optimization using evolutionary algorithms—a comparative case study. In Parallel problem solving from nature—PPSN V (pp. 292-301). Springer Berlin Heidelberg.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. Evolutionary Computation, IEEE Transactions on, 7(2), 117-132.

4 CONCLUSÃO GERAL

Esta pesquisa contemplou o problema de sequenciamento de tarefas em um ambiente *assembly flowshop* com três estágios (3sAFS) e tempos de preparação dependentes da sequência. Este problema é conhecido como NP-difícil e é comumente encontrado em indústrias de manufatura. Para sua resolução foram realizadas duas abordagens. A primeira abordagem propõe uma otimização mono-objetivo para minimização do atraso total, enquanto a segunda propõe uma otimização multi-objetivo, através da geração de um conjunto de soluções Pareto (soluções não dominadas), para minimização do fluxo total e atraso total.

Ambas as abordagens obtiveram sucesso na resolução do problema. Na abordagem mono-objetivo, todos os algoritmos heurísticos propostos obtiveram melhores resultados que o algoritmo da literatura PSA, para instâncias de pequeno e grande porte. Os resultados foram validados por diversas análises estatísticas. Na abordagem multi-objetivo, foi mostrado que o uso da metaheurística *Iterated greedy*, para realização de uma intensificação gulosa dentro do tradicional algoritmo de resolução de problemas multi-objetivo, NSGA-II, aumenta consideravelmente a qualidade das soluções Pareto encontradas. Além disso, também é proposto uma busca local em descida dentro do processo de intensificação. Os resultados das análises estatísticas apontam uma melhora ainda maior na qualidade das soluções Pareto.

Sendo assim, é comprovado que a hipótese (levantada no início da pesquisa), que o problema 3sAFS pode ser resolvido com metaheurísticas, é verdadeira. Também, é perceptível que o objetivo geral considerado para este trabalho é cumprido. Da mesma maneira, os objetivos específicos são alcançados ao longo do desenvolvimento do mesmo.

O presente trabalho representa um avanço na resolução do 3sAFS, contribuindo para aumento da literatura especializada. Porém, o problema ainda não está completamente saturado. Há espaço para a realização de novas abordagens, conforme sugerido na seção de trabalhos futuros, existente no final de cada artigo.

Além disso, o conteúdo teórico e prático apresentado neste trabalho pode ser utilizado na resolução de outros problemas de otimização e sequenciamento de tarefas.

APENDICE A

DADOS DO EXPERIMENTO COMPUTACIONAL PARA COMPARAÇÃO DOS ALGORITMOS ILS, IG, ILS-IG, GRASP-RVND E PSA EM INSTÂNCIAS DE PEQUENO PORTE

Esta seção mostra os resultados obtidos na segunda etapa do experimento computacional de comparação dos algoritmos propostos para resolução do 3sAFS. Nesta etapa, os algoritmos foram rodados sobre as instâncias de pequeno porte e comparados com resultado ótimo, obtido através da resolução do modelo matemático no CPLEX. O CPLEX foi configurado para retornar o melhor resultado alcançado em até 2 horas, mas todos os resultados ótimos foram encontrados antes deste tempo. Porém, para a maioria das instâncias de 14 tarefas, o CPLEX retornou erro por estouro de memória. Assim, apenas duas instâncias com este tamanho foram resolvidas.

Através da Tabela 39, é possível visualizar os resultados obtidos por todos os algoritmos em todas as instâncias de pequeno porte, juntamente com o resultado ótimo encontrado através da resolução do modelo. A coluna "Id" mostra o identificador da instância, mostrando que são vários problemas diferentes. A coluna "Tamanho" mostra a combinação $n \times m$ da instância. A coluna "Melhor solução conhecida" se faz necessária, pois algumas instâncias (com 14 tarefas) não puderam ser resolvidas pelo modelo. Assim, para estes casos, o valor apresentado nesta coluna corresponde à melhor solução encontrada pelos algoritmos heurísticos. As colunas "Resultado ótimo" e "Tempo" mostram, respectivamente, o resultado ótimo para a instância e o tempo gasto para sua resolução no CPLEX. A coluna "Quant. ótimos encontrados" mostra a quantidade de vezes que o algoritmo encontrou a solução ótima para a instância, em um total de 10 execuções para cada instância. As colunas seguintes "RPD médio" e "Tempo médio" mostram, respectivamente, a distância média (desvio relativo padrão médio) entre o resultado retornado pelo algoritmo e a melhor solução conhecida e o tempo médio gasto para obter o resultado.

Tabela 39: Resultado do experimento de comparação dos algoritmos e modelo matemático para instâncias de tamanho pequeno

Instância			Modelo matemático		PSA			ILS			IG			ILS-IG			GRASP-RVND		
Id.	Tamanho (n x m)	Melhor Solução Conhecida	Resultado ótimo	Tempo (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)
1	08 X 02	1097	1097	1,50	10 / 10	0,00	1,57	10 / 10	0,00	1,60	0 / 10	4,65	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,62
2	08 X 02	677	677	1,61	10 / 10	0,00	1,57	10 / 10	0,00	1,60	10 / 10	0,00	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,62
3	08 X 02	1085	1085	1,77	0 / 10	3,55	1,56	10 / 10	0,00	1,60	10 / 10	0,00	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,61
4	08 X 02	883	883	1,83	10 / 10	0,00	1,56	10 / 10	0,00	1,60	0 / 10	15,52	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,61
5	08 X 02	1169	1169	1,56	9 / 10	0,33	1,56	10 / 10	0,00	1,60	10 / 10	0,00	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,61
6	08 X 02	1437	1437	1,56	0 / 10	1,10	1,57	10 / 10	0,00	1,60	10 / 10	0,00	1,61	0 / 10	1,04	1,61	10 / 10	0,00	1,62
7	08 X 02	892	892	1,84	10 / 10	0,00	1,57	10 / 10	0,00	1,60	0 / 10	6,73	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,61
8	08 X 02	1209	1209	1,61	10 / 10	0,00	1,56	10 / 10	0,00	1,60	0 / 10	11,17	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,62
9	08 X 02	1457	1457	1,97	0 / 10	1,03	3,47	10 / 10	0,00	1,60	10 / 10	0,00	1,61	0 / 10	1,03	1,61	10 / 10	0,00	1,62
10	08 X 02	1776	1776	1,91	10 / 10	0,00	3,91	10 / 10	0,00	1,60	0 / 10	8,05	1,61	10 / 10	0,00	1,61	10 / 10	0,00	1,61
11	08 X 04	889	889	1,78	10 / 10	0,00	3,82	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,22
12	08 X 04	1389	1389	2,41	0 / 10	0,35	3,70	10 / 10	0,00	3,20	0 / 10	14,76	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,22
13	08 X 04	2083	2083	1,75	0 / 10	1,76	3,08	10 / 10	0,00	3,20	0 / 10	0,05	3,21	0 / 10	0,05	3,21	10 / 10	0,00	3,22
14	08 X 04	1318	1318	1,99	5 / 10	0,61	3,08	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,21
15	08 X 04	1370	1370	1,52	0 / 10	2,34	3,08	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,22
16	08 X 04	1266	1266	1,59	8 / 10	0,92	3,07	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,22
17	08 X 04	1118	1118	2,28	0 / 10	1,61	3,09	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,21
18	08 X 04	747	747	1,66	10 / 10	0,00	3,07	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,22
19	08 X 04	1212	1212	3,31	10 / 10	0,00	3,07	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,21
20	08 X 04	1130	1130	2,19	8 / 10	0,25	3,07	10 / 10	0,00	3,20	10 / 10	0,00	3,21	10 / 10	0,00	3,21	10 / 10	0,00	3,21
21	08 X 06	1180	1180	2,13	9 / 10	0,35	4,55	10 / 10	0,00	4,80	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
22	08 X 06	1227	1227	1,95	0 / 10	4,01	4,54	10 / 10	0,00	4,80	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
23	08 X 06	567	567	1,78	0 / 10	21,73	4,54	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
24	08 X 06	926	926	1,78	10 / 10	0,00	4,55	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
25	08 X 06	1068	1068	2,19	10 / 10	0,00	5,74	10 / 10	0,00	4,81	0 / 10	1,03	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
26	08 X 06	1243	1243	1,75	10 / 10	0,00	7,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
27	08 X 06	1135	1135	2,17	1 / 10	3,01	7,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
28	08 X 06	1198	1198	1,92	2 / 10	7,08	6,59	10 / 10	0,00	4,81	0 / 10	7,60	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
29	08 X 06	1005	1005	2,41	0 / 10	5,57	5,04	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
30	08 X 06	613	613	2,70	10 / 10	0,00	4,54	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
31	10 X 02	1038	1038	30,10	6 / 10	0,63	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
32	10 X 02	1502	1502	15,58	8 / 10	0,21	1,94	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
33	10 X 02	1161	1161	16,77	10 / 10	0,00	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,01

Instância		Melhor Solução Conhecida	Modelo matemático		PSA			ILS			IG			ILS-IG			GRASP-RVND		
Id.	Tamanho (n x m)		Resultado ótimo	Tempo (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)
34	10 X 02	1396	1396	5,36	9 / 10	0,16	1,96	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
35	10 X 02	1105	1105	10,44	0 / 10	7,49	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
36	10 X 02	1599	1599	4,94	7 / 10	0,26	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
37	10 X 02	1628	1628	11,35	10 / 10	0,00	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
38	10 X 02	1315	1315	12,89	10 / 10	0,00	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,01
39	10 X 02	2160	2160	4,27	0 / 10	4,54	1,95	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
40	10 X 02	1331	1331	22,46	1 / 10	12,26	1,96	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00	10 / 10	0,00	2,00
41	10 X 04	1840	1840	17,33	6 / 10	1,14	3,83	10 / 10	0,00	4,00	3 / 10	0,46	4,00	10 / 10	0,00	4,01	10 / 10	0,00	4,01
42	10 X 04	863	863	15,30	8 / 10	2,71	3,83	10 / 10	0,00	4,00	10 / 10	0,00	4,00	10 / 10	0,00	4,00	10 / 10	0,00	4,00
43	10 X 04	1301	1301	14,53	0 / 10	3,77	3,83	10 / 10	0,00	4,00	10 / 10	0,00	4,00	10 / 10	0,00	4,00	10 / 10	0,00	4,01
44	10 X 04	2247	2247	16,50	2 / 10	0,89	3,82	10 / 10	0,00	4,00	0 / 10	0,18	4,01	10 / 10	0,00	4,01	10 / 10	0,00	4,01
45	10 X 04	1202	1202	5,74	0 / 10	4,03	3,82	10 / 10	0,00	4,00	10 / 10	0,00	4,01	10 / 10	0,00	4,00	10 / 10	0,00	4,01
46	10 X 04	1329	1329	6,25	0 / 10	4,15	3,83	10 / 10	0,00	4,00	10 / 10	0,00	4,01	10 / 10	0,00	4,00	10 / 10	0,00	4,01
47	10 X 04	1959	1959	24,35	3 / 10	2,16	3,82	10 / 10	0,00	4,00	10 / 10	0,00	4,01	10 / 10	0,00	4,00	10 / 10	0,00	4,01
48	10 X 04	1276	1276	9,81	1 / 10	14,08	3,83	10 / 10	0,00	4,00	0 / 10	16,22	4,01	10 / 10	0,00	4,01	10 / 10	0,00	4,01
49	10 X 04	1573	1573	14,91	10 / 10	0,00	3,83	10 / 10	0,00	4,00	10 / 10	0,00	4,01	10 / 10	0,00	4,00	10 / 10	0,00	4,01
50	10 X 04	1238	1238	13,47	6 / 10	1,62	3,81	10 / 10	0,00	4,00	10 / 10	0,00	4,00	10 / 10	0,00	4,00	10 / 10	0,00	4,01
51	10 X 06	1876	1876	31,32	2 / 10	3,37	5,66	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
52	10 X 06	1357	1357	22,91	0 / 10	4,04	5,65	10 / 10	0,00	6,01	5 / 10	1,51	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
53	10 X 06	1390	1390	13,24	10 / 10	0,00	5,66	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
54	10 X 06	1591	1591	13,72	0 / 10	5,00	5,66	10 / 10	0,00	6,01	0 / 10	3,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
55	10 X 06	2021	2021	18,36	0 / 10	1,58	5,67	10 / 10	0,00	6,01	0 / 10	1,58	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
56	10 X 06	2076	2076	15,22	0 / 10	5,40	5,65	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
57	10 X 06	1822	1822	14,51	8 / 10	0,29	5,66	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
58	10 X 06	1503	1503	11,75	0 / 10	5,87	5,66	10 / 10	0,00	6,01	0 / 10	26,35	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,02
59	10 X 06	2137	2137	12,41	1 / 10	2,62	5,64	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
60	10 X 06	1335	1335	6,08	10 / 10	0,00	5,67	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01	10 / 10	0,00	6,01
61	12 X 02	2583	2583	31,19	0 / 10	3,38	2,34	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
62	12 X 02	1860	1860	34,26	0 / 10	2,26	2,32	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
63	12 X 02	2256	2256	197,07	1 / 10	0,93	2,34	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
64	12 X 02	1875	1875	59,29	2 / 10	1,30	2,33	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
65	12 X 02	1500	1500	201,24	0 / 10	4,55	2,33	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
66	12 X 02	1854	1854	47,44	0 / 10	3,98	2,33	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
67	12 X 02	1620	1620	660,87	7 / 10	0,71	2,34	10 / 10	0,00	2,41	0 / 10	2,47	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41

Instância		Melhor Solução Conhecida	Modelo matemático		PSA			ILS			IG			ILS-IG			GRASP-RVND		
Id.	Tamanho (n x m)		Resultado ótimo	Tempo (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)
68	12 X 02	2328	2328	99,42	10 / 10	0,00	2,33	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
69	12 X 02	2425	2425	59,17	3 / 10	4,10	2,33	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
70	12 X 02	2060	2060	2119,00	7 / 10	0,27	2,34	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41	10 / 10	0,00	2,41
71	12 X 04	1918	1918	74,79	0 / 10	7,66	4,57	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
72	12 X 04	1919	1919	1961,33	3 / 10	0,72	4,59	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
73	12 X 04	2377	2377	440,81	0 / 10	5,15	4,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
74	12 X 04	2174	2174	76,73	2 / 10	1,39	4,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
75	12 X 04	1669	1669	31,94	0 / 10	3,40	4,59	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
76	12 X 04	1761	1761	219,90	3 / 10	1,52	4,58	10 / 10	0,00	4,81	0 / 10	8,29	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
77	12 X 04	1542	1542	315,95	0 / 10	6,42	4,57	10 / 10	0,00	4,81	0 / 10	17,51	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
78	12 X 04	1394	1394	369,35	0 / 10	10,98	4,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
79	12 X 04	978	978	1402,82	0 / 10	5,14	4,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81
80	12 X 04	1656	1656	1625,94	7 / 10	0,92	4,58	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,81	10 / 10	0,00	4,82
81	12 X 06	2759	2759	188,02	1 / 10	2,61	6,77	10 / 10	0,00	7,20	0 / 10	1,45	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,21
82	12 X 06	2487	2487	30,90	4 / 10	1,13	6,77	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20
83	12 X 06	2160	2160	171,39	2 / 10	3,13	6,77	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,21
84	12 X 06	2242	2242	735,79	3 / 10	1,59	6,76	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,21
85	12 X 06	2411	2411	137,56	5 / 10	0,75	6,76	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,21
86	12 X 06	1834	1834	5841,22	0 / 10	5,67	6,78	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20
87	12 X 06	2553	2553	229,32	8 / 10	1,32	6,77	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20
88	12 X 06	2231	2231	151,17	0 / 10	4,72	6,76	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,21
89	12 X 06	2034	2034	53,13	6 / 10	0,61	6,75	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20
90	12 X 06	1876	1876	1722,32	5 / 10	0,84	6,76	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,20	10 / 10	0,00	7,21
91	14 X 02	1634	1634	3801,66	0 / 10	6,89	2,73	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
92	14 X 02	1911	1911	3632,44	5 / 10	0,88	2,71	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
93	14 X 02	1992	2123	7200,6	9 / 10	0,78	2,71	10 / 10	0,00	2,81	0 / 10	3,26	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
94	14 X 02	2768	2768	1769,02	0 / 10	0,87	2,71	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
95	14 X 02	2427	2427	935,925	2 / 10	4,07	2,70	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
96	14 X 02	2596	2671	7203,01	0 / 10	1,03	2,69	10 / 10	0,00	2,81	4 / 10	0,59	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
97	14 X 02	2238	2238	7155,73	1 / 10	1,74	2,71	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
98	14 X 02	3216	3216	3693,29	2 / 10	2,04	2,72	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
99	14 X 02	1852	1852	152,735	0 / 10	2,81	2,71	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
100	14 X 02	2293	2293	135,089	2 / 10	0,70	2,71	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81	10 / 10	0,00	2,81
101	14 X 04	2286	2389	7201,45	0 / 10	2,36	5,30	10 / 10	0,00	5,61	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60

Instância		Melhor Solução Conhecida	Modelo matemático		PSA			ILS			IG			ILS-IG			GRASP-RVND		
Id.	Tamanho (n x m)		Resultado ótimo	Tempo (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)	Quant. ótimos encontrados	RPD Médio	Tempo Médio (seg.)
102	14 X 04	1552	1643	7201,43	0 / 10	6,97	5,32	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,61
103	14 X 04	2599	2599	2194,72	0 / 10	1,35	5,31	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60
104	14 X 04	1888	1987	7203,53	8 / 10	1,40	5,31	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60
105	14 X 04	3327	3327	2535,51	10 / 10	0,00	5,31	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,61
106	14 X 04	2495	2597	7200,86	0 / 10	7,09	5,31	10 / 10	0,00	5,60	0 / 10	0,60	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60
107	14 X 04	2185	2185	3433,79	0 / 10	1,97	5,30	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60
108	14 X 04	2720	2720	5587,42	0 / 10	2,46	5,31	10 / 10	0,00	5,60	2 / 10	1,06	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,61
109	14 X 04	2001	2001	1649,64	0 / 10	2,89	5,31	10 / 10	0,00	5,60	7 / 10	1,06	5,60	10 / 10	0,00	5,60	10 / 10	0,00	5,60
110	14 X 04	2102	2396	7201,02	0 / 10	13,31	5,29	10 / 10	0,00	5,61	10 / 10	0,00	5,60	10 / 10	0,00	5,61	10 / 10	0,00	5,61
111	14 X 06	2887	2887	4434,11	2 / 10	4,09	8,22	10 / 10	0,00	8,41	4 / 10	2,39	8,41	10 / 10	0,00	8,40	10 / 10	0,00	8,42
112	14 X 06	3629	3629	2276,93	3 / 10	1,87	8,22	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,40	10 / 10	0,00	8,41
113	14 X 06	1367	1401	7201,28	10 / 10	0,00	8,22	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,40	10 / 10	0,00	8,42
114	14 X 06	2294	2294	2855,43	5 / 10	0,75	8,22	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,40	10 / 10	0,00	8,41
115	14 X 06	2148	2239	7201,15	4 / 10	1,36	8,23	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,40	10 / 10	0,00	8,41
116	14 X 06	2773	2842	7202,73	0 / 10	0,50	8,22	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,42
117	14 X 06	2802	2802	999,506	10 / 10	0,00	8,22	10 / 10	0,00	8,41	0 / 10	0,86	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41
118	14 X 06	2486	2486	5035,71	5 / 10	1,05	8,23	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41
119	14 X 06	2955	2955	5497,02	0 / 10	2,63	8,22	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41
120	14 X 06	4186	4262	7201,86	2 / 10	1,38	8,23	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,41	10 / 10	0,00	8,42