VINÍCIUS VILAR JACOB

APLICAÇÃO DE METAHEURÍSTICAS PARA PROBLEMAS DE SEQUENCIAMENTO COM LOTES DE TAREFAS

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 Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
J15a
2014

Jacob, Vinícius Vilar, 1987-
Aplicação de metaheurísticas para problemas de sequenciamento com lotes de tarefas / Vinícius Vilar Jacob. – Viçosa, MG, 2014.
xv, 149f. : il. (algumas color.) ; 29 cm.

Inclui apêndices.
Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.114-121.

1. Algorítmos. 2. Programação heurística. 3. Programação (Matemática). I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22. ed. 518.1

VINÍCIUS VILAR JACOB

APLICAÇÃO DE METAHEURÍSTICAS PARA PROBLEMAS DE SEQUENCIAMENTO COM LOTES DE TAREFAS

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: 04 de julho de 2014.

Luciana Brugiolo Gonçalves

André Gustavo dos Santos
Coorientador

Haroldo Gambini Santos

José Elias Claudio Arroyo
Orientador

Aos meus amados pais, Vicente e Eliane.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela saúde, paciência e força para concluir esta difícil etapa. Quando nada dava certo era a Ele que eu recorria.

Agradeço especialmente aos meus pais, Vicente e Eliane, pelo apoio incondicional e pela educação que me deram. Tenho certeza de que sem o incentivo e ajuda dos dois nada disso seria possível. Vocês são exemplos de caráter e personalidade. Tenho orgulho de vocês.

Agradeço a minha irmã Amanda que me deu o amor da minha vida, Millena.

A todos os meus familiares pela palavras de incentivo. A todos os meus amigos pelos momentos de diversão e descontração.

Agradeço também a minha namorada Júnia, pelo carinho, pela dedicação e principalmente pela paciência. Tenho absoluta certeza de que este período também não foi fácil pra você. Obrigado pelo seu companheirismo e amizade.

A todos os professores e funcionários do Departamento de Informática por auxiliarem na minha formação. Tenho certeza que levo boas lembranças de cada um de vocês. Especialmente, aos meus orientadores, José Elias e André Gustavo, e a professora Luciana, pela atenção e pelos ensinamentos. Aprendi muito com vocês. Não poderia deixar de agradecer ao Harlem Madrid e Flávio Maia pela grande contribuição que derem ao meu trabalho. À Diretoria de Tecnologia da Informação da Universidade Federal de Viçosa pela infraestrutura disponibilizada.

Obrigado a todos vocês.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

JACOB, Vinícius Vilar, M.Sc., Universidade Federal de Viçosa, Julho de 2014. **Aplicação de metaheurísticas para problemas de sequenciamento com lotes de tarefas** Orientador: José Elias Claudio Arroyo.

Diversos processos industriais podem ser modelados através de problemas de sequenciamento. A otimização destes processos, por sua vez, é de grande interesse das indústrias já que pode acarretar em aumento da produtividade e lucratividade das mesmas. Algumas vezes, técnicas de loteamento podem ser aplicadas para melhorar a utilização dos recursos de produção, levando a ganhos em eficiência no processo produtivo. Neste trabalho, são abordados dois problemas complexos de sequenciamento da produção que envolvem considerações de formação de lotes. No primeiro, a formação de um lote é condicionada ao processamento de diversas tarefas com características semelhantes, ditas serem da mesma família. Neste caso, não há necessidade de tempos de preparação entre tarefas do mesmo lote. No outro problema, a formação de um lote está associada aos custos de entrega, sendo que lotes com muitas tarefas implicam em baixos custos de entrega. No primeiro caso é tratado o problema de sequenciamento em uma máquina com tempos de preparação dependentes da sequência das famílias das tarefas e a minimização do atraso total. No segundo, é tratado o problema de sequenciamento em uma máquina e a minimização do fluxo ponderado total mais os custos de entrega dos lotes.
O objetivo desta dissertação é apresentar, discutir e tratar estes dois problemas aplicando principalmente heurísticas e comparar os resultados com aqueles disponíveis na literatura. Para o primeiro problema, foram propostas três heurísticas baseadas na metaheurística Busca Local Iterada (ILS) e para o segundo foi proposta uma heurística baseada em ILS e outra baseada na metaheurística *Iterated Greedy* (IG), além de um modelo de Programação Linear Inteira Mista (MILP).

Abstract

JACOB, Vinícius Vilar, M.Sc., Universidade Federal de Viçosa, July of 2014. **Application of metaheuristics for scheduling problems with batching.** Adviser: José Elias Claudio Arroyo.

Many manufacturing processes can be modeled as a scheduling problem. The optimization of these processes is of great interest to the industry since it can result in higher productivity and profitability. Sometimes, batching can be applied to improve the utilization of production resources, leading to efficiency gains in the production process. In this paper, two complex scheduling problems involving considerations of batching are addressed. In the first, the formation of a batch is conditioned to processing various jobs of the same family consecutively. In this case, there is no setup time between jobs of same batch (family). In the other problem, the batch formation is associated with delivery costs, where batches with many jobs implies low costs of delivery. In the first problem is addressed the single machine scheduling problem with sequence dependent setup times and total tardiness minimization. In the second, the problem addressed is the single machine scheduling problem and minimizing of the total weighted flow time and delivery costs.
The aim of this dissertation is to present, discuss and solve these two scheduling problems applying heuristics and to compare the results with those available in the literature. We proposed three heuristics based on metaheuristic Iterated Local Search (ILS) for the first problem. We also proposed a heuristic based on ILS and other based on metaheuristic Iterated Greedy (IG) and a Mixed Integer Linear Programming model (MILP) for the second problem.

Capítulo 1

Introdução

1.1 Considerações iniciais

A Programação da Produção teve seu início junto a Pesquisa Operacional no início do século XX, sendo que os problemas associados a operações militares eram o seu principal foco. Um dos principais trabalhos desta época remete a Henry Gantt, que foi um dos pioneiros a desenvolver um sistema da programação da produção baseado em gráficos e cálculos, bem como considerar restrições de capacidade e tempo (Lustosa et al., 2008).

Segundo Pinedo (2012), a programação da produção "*é um processo de tomada de decisões que é usado na base regular de muitas indústrias de manufaturas e serviços. Ela lida com alocação de recursos e tarefas sobre períodos de tempo*". Problemas da programação da produção têm sido amplamente estudados na literatura desde os anos 50. Normalmente 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 critérios simultaneamente, os quais podem estar associados a utilização eficiente dos recursos, diminuição do tempo total da produção, atendimento às datas combinadas com os clientes para entrega dos produtos, diminuição dos custos de produção, de estoques e de entregas, dentre outros.

A programação da produção é fundamentalmente um processo de otimização e, por isso, é de grande interesse das indústrias de manufatura já que as empresas querem aumentar a sua produtividade e lucratividade (Shen et al., 2006). A melhoria de um sistema de produção pode ser alcançada adquirindo-se novos equipamentos (máquinas), mas há casos onde é inviável que isto seja feito, como na

fabricação de componentes eletrônicos, onde os preços das máquinas são extremamente altos e desta forma é desejável que se utilize de forma adequada os recursos de produção já existentes (Mathirajan & Sivakumar, 2006).

Técnicas de loteamento (*batching*) podem ser aplicadas para melhor utilização dos recursos de produção, levando a ganhos em eficiência na produção. Por eficiência pode-se entender *economia* financeira ou *diminuição* do tempo necessário na produção.

Por isso, há um interesse prático em estudar problemas que envolvem loteamento, visto que, mediante a globalização e ampla concorrência enfrentadas pelas empresas, a utilização eficiente dos recursos disponíveis pode ser um diferencial e fornecer vantagens competitivas. Além disso, como muitos outros problemas de otimização combinatória, existe também um grande interesse teórico visto que são problemas desafiadores e difíceis de resolver. Muitos pertencem a classe de problemas $\mathcal{NP}$-Difícil. Isso significa que não se conhece algoritmos eficientes que possam resolvê-los de forma exata em um tempo computacional aceitável para aplicações práticas. Logo, é adequado a aplicação de técnicas heurísticas para resolvê-los de forma aproximada em tempo um aceitável.

1.2 O problema e sua importância

Como mencionado anteriormente, técnicas de loteamento na programação da produção são utilizadas com o foco em eficiência na produção tanto na economia dos recursos financeiros gastos quanto na diminuição do tempo necessário na produção. Existem algumas situações onde é mais rápido e mais barato processar tarefas em lotes do que processá-las individualmente. Um exemplo dessa situação é dado por Potts & Kovalyov (2000) onde há benefícios que podem resultar do loteamento de tarefas quando máquinas requerem algum tempo de preparação (ou *setup time* ou, ainda, simplesmente *setup*) para o processamento de tarefas que possuem diferentes características.

Este tempo de preparação pode ser necessário para obter ferramentas, limpeza de máquina, posicionamento de acessórios e inspeção de materiais antes que uma tarefa possa começar o seu processamento. Neste caso, denominado *modelo baseado em famílias*, as tarefas são particionadas em famílias de acordo com sua similaridade.

Uma situação prática ocorre na fabricação de tubos de aço de diferentes especificações, onde tubos com o mesmo diâmetro podem ser considerados serem da mesma família. Quando trocam-se as especificações dos diâmetros dos tubos a serem

produzidos deve-se configurar a máquina para tal requerimento. Ou seja, é necessário um tempo para preparar a máquina sempre que troca-se o processamento de tarefas de uma família para outra. Entre as tarefas da mesma família, que compartilham características em comum, não é necessário nenhum tempo de preparação.

No modelo baseado em famílias, um lote é um conjunto de tarefas que estão sequenciadas contiguamente em uma mesma máquina. Entre as tarefas do mesmo lote não é preciso um tempo de preparação de máquina. Alta utilização da máquina é alcançada quando se considera lotes com tarefas da mesma família porque um menor número de *setups* será necessário. Por outro lado, processar grandes lotes pode atrasar o processamento de tarefas importantes de outras famílias comprometendo a programação da produção como um todo (Potts & Kovalyov, 2000).

Outra situação onde pode haver melhoramento da eficiência ao se agrupar tarefas em lotes ocorrem quando uma máquina possui a capacidade de processar um conjunto de tarefas simultaneamente. Estas máquinas são ditas serem de *processamento em lote*. Neste caso, um lote é o próprio conjunto de tarefas que são processadas de uma única vez na máquina. Problemas deste tipo ocorrem na fabricação de placas de circuitos integrado, na operação de impressão, onde várias tarefas são colocadas simultaneamente em fornos. Podem ocorrer também em processos químicos que acontecem em tanques (Potts & Kovalyov, 2000).

Ainda, segundo Potts & Kovalyov (2000), os modelos baseados em família podem ser caracterizados quanto à disponibilidade das tarefas após o seu processamento. No primeiro caso, dito ser de disponibilidade de tarefas (*job availability*), as tarefas estão disponíveis individualmente, para serem entregues aos clientes ou para a próxima estação de processamento, logo após terem sido processadas. O outro caso, dito ser disponibilidade de lote (*batch availability*), as tarefas que estão agrupadas em um mesmo lote estão disponíveis somente após todas as outras do mesmo lote terem sido processadas.

Considerando o modelo baseado em famílias com disponibilidade de tarefa, o tempo de preparação costuma ser caracterizado como *independente* ou *dependente* da sequência da família das tarefas. No modelo independente, o *setup* é determinado somente pela próxima tarefa que será processada. Quando é dependente da sequência das famílias, o *setup* depende tanto da tarefa recém processada quando da tarefa seguinte.

No modelo que considera a disponibilidade de lotes, às vezes, é importante considerar os custos associados à entrega dos lotes. Estes casos, onde os custos de entrega são levados em conta, são conhecidos como *batch delivery system*. Um exemplo prático desta situação ocorre se tarefas em um mesmo lote, após terem sido

processadas, são colocadas em um palete, que é movido a partir da máquina para outra estação de processamento ou para serem entregues aos clientes, quando todas as tarefas do lote tenham sido processadas.

É importante destacar que o modelo baseado em famílias com disponibilidade de lotes difere do caso onde a máquina possui a capacidade de processamento em lote. No primeiro caso, as tarefas são processadas sequencialmente tal que o tempo de processamento do lote é igual a soma do tempo de processamento de todas as tarefas que estão no lote e o tempo de conclusão das tarefas é o tempo de conclusão do lote, que é definido pela última tarefa processada do lote. No segundo caso, todas as tarefas iniciam e terminam o seu processamento juntas, sendo que o tempo de processamento do lote é determinado pela tarefa que possui o maior tempo de processamento.

Um fator que pode aumentar a complexidade deste problemas é tratar explicitamente as datas de liberação (*release time*) das tarefas. Isto é de grande importância do ponto de vista prático, já que envolve o caso onde alguns produtos (ou componentes) que são requeridos para o processamento ainda não chegaram à fábrica. Em alguns problemas da programação da produção que envolvem a formação de lotes para processamento simultâneo (como ocorre na manufatura de componentes eletrônicos (Mathirajan & Sivakumar, 2006)) a data de liberação é importante visto que a formação do lote só pode ocorrer depois que todas as tarefas estejam liberadas.

Neste trabalho, serão abordados dois problemas da programação da produção que envolvem o conceito de lotes. O primeiro considera o modelo baseado em famílias com disponibilidade de tarefa e é denotado por $1|ST_{sd,b}|\sum T_j$, onde 1 representa o ambiente de uma única máquina, $ST_{sd,b}$ informação do tempo de preparação como dependente da sequência das famílias e $\sum T_j$ é relativo ao critério de otimalidade tratado, que é minimizar o atraso total. Este problema considera tempo de preparação dependente das famílias e ocorre em algumas situações práticas, como na linha de produção de plásticos coloridos, onde o tempo de preparação existe quando se troca de uma cor para outra (Potts & Van Wassenhove, 1992) e na manufatura de tubos de aço, onde o tempo de preparação existe ao fabricar tubos de diferentes padrões ou especificações (Byong-Hun & Jae-Ho, 1990). Em todos os casos, o tempo de preparação é um gargalo na linha de produção e deve ser tratado de forma adequada. Mais detalhes sobre este problema serão dados na Parte I.

O segundo problema considera o modelo baseado em famílias com disponibilidade de lotes e custos de entrega. Ele é denotado por $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, onde 1 representa o ambiente de uma única máquina, r_{ij} informação de que cada

tarefa possui uma data de liberação e $\sum w_{ij}F_{ij} + \sum \delta_i d_i$ é relativo ao critério de otimalidade tratado, em que $\sum w_{ij}F_{ij}$ é a parcela correspondente ao tempo total de fluxo ponderado e $\sum \delta_i d_i$ é o custo total de entrega (onde, para um cliente i, δ_i é o número lotes entregues a ele e d_i é o custo de entrega de cada lote para este cliente). Este problema tem recebido bastante atenção em pesquisas de logística e gerenciamento da produção. Ocorre na cadeia de fornecimento de dois estágios, quando existe a necessidade da integração entre programação da produção e transporte: quando produtos semimanufaturados precisam ser transportados da área de estoque para outro futuro processamento. Esta integração pode ajudar as indústrias a economizarem energia e reduzir o consumo de combustível (Hamidinia et al., 2012; Haddad et al., 2012; Mazdeha et al., 2011; Rasti-Barzoki et al., 2012). Mais detalhes sobre este problema serão dados na Parte II.

Além de motivação prática, os dois problemas alvos deste trabalho são também interessantes do ponto de vista de complexidade computacional, já que são problemas classificados como $\mathcal{NP}$-Difíceis (Du & Leung, 1990; Ji et al., 2007). São problemas desafiadores já que não se conhece algoritmos eficientes para sua resolução.

1.3 Hipótese

Os dois problemas em questão são problemas de otimização combinatória considerados de difícil solução. Para encontrar soluções para este tipo de problema, em tempo aceitável, geralmente devem ser utilizados métodos heurísticos. Este trabalho baseia-se na hipótese de que, para estes problemas, é possível aplicar heurísticas, baseadas em metaheurísticas, de modo que se encontre soluções aproximadas de "boa" qualidade em um tempo computacional aceitável.

Um algoritmo *Branch and Bound*, descrito na literatura para o problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, será usado para comparação. Para o outro problema, $1|ST_{sd,b}|\sum T_j$, um modelo matemático de Programação Linear Inteira Mista e um Algoritmo Genético descritos na literatura serão usados para comparação.

1.4 Objetivos

O objetivo geral do projeto é desenvolver heurísticas, baseadas em metaheurística, que melhorem os resultados de algoritmos encontrados na literatura. Melhorar os resultados pode ser: encontrar as mesmas soluções em tempo computacional me-

nor; ou encontrar soluções de melhor qualidade; ou ainda resolver problemas com dimensões maiores.

Especificamente, pretende-se realizar os seguintes passos:

(a) Obter, ou gerar, os problemas testes (instâncias) utilizados na literatura para serem utilizados nos experimentos

(b) Obter, ou implementar, os algoritmos já propostos na literatura

(c) Definir um esquema de representação de solução que possa ser utilizado em uma metaheurística e num procedimento de busca local

(d) Propor e implementar as heurísticas

(e) Testar as implementações

(f) Comparar os resultados utilizando métodos estatísticos

1.5 Referencial teórico

1.5.1 Problemas de otimização combinatória

De forma geral, em um problema de otimização combinatória busca-se encontrar uma solução cujo valor associado seja ótimo (mínimo ou máximo), dentro de um conjunto discreto de soluções possíveis.

Um conceito importante é o de *solução viável*: uma solução é dita ser viável se ela respeita todas as restrições do problema. Segundo Talbi (2009), um problema de otimização pode ser definido por uma tupla (S, f), onde S representa um conjunto de soluções viáveis (também comumente chamado de espaço de soluções) e f é uma função, chamada de *função objetivo* $f : S \rightarrow \mathbb{R}$. A função objetivo f associa a cada solução $s \in S$ do *espaço de busca* um número real indicando a sua qualidade, de modo que exista um relacionamento entre as soluções. O objetivo de um problema de otimização é encontrar uma solução $s^* \in S$ que seja um ***ótimo global***, onde:

Definição 1 ***Ótimo global.*** *Para um problema de minimização, uma solução $s^* \in S$ é um ótimo global se ela tem uma melhor função objetivo de todas as soluções dos espaço de busca, isto é, $\forall s \in S$, $f(s^*) \leq f(s)$* (Talbi, 2009).

Para um problema de minimização, o ótimo global é aquela solução que possui o menor valor da função objetivo e, para um problema de maximização, o ótimo global é a solução com o maior valor da função objetivo.

Outro conceito importante é o de *vizinhança*. Duas soluções são ditas vizinhas se, a partir de determinada alteração (movimento) em uma solução, se alcança a outra. Logo, tem-se o conceito de *ótimos locais* com respeito a uma vizinhança:

Definição 2 ***Ótimo Local.*** *Uma solução $s \in S$ é dita ser um ótimo local se ela tem a melhor função objetivo dentre todas as soluções que são vizinhas. Isto é, $\forall s' \in N(s)$, $f(s) \leq f(s')$* (Talbi, 2009)*, onde $N(s)$ são os vizinhos de s.*

Assim, um *ótimo global é necessariamente o melhor dentre todos os ótimos locais.*

1.5.2 Metaheurísticas

Para diversos problemas complexos, as metaheurísticas fornecem "boas" soluções em um tempo computacional razoável. Ao contrário dos métodos de otimização exatos, as metaheurísticas não garantem a otimalidade das soluções obtidas e não definem quão perto estão da solução ótima.

Metaheurísticas são *frameworks* que definem procedimentos específicos para resolver uma ampla gama de problemas de otimização combinatória. Sua estrutura é descrita através de componentes genéricos, que devem então ser adaptados ao problema que se quer resolver. Elas exploram tanto o conceito de *intensificação* como *diversificação*. A diversificação refere-se a exploração do espaço de busca e intensificação à melhoria iterativa da solução. Algumas propriedades fundamentais caracterizam uma metaheurística (Blum & Roli, 2003):

- São estratégias que "guiam" o processo de busca
- O objetivo é explorar o espaço de busca para encontrar soluções ótimas (ótimo global) ou soluções aproximadas
- São algoritmos aproximados e, muitas vezes, não determinísticos
- Possuem mecanismos para escapar de ótimos locais
- Podem utilizar conhecimento específico do domínio e experiência acumulada para "guiar" a busca

Existe uma variedade muito grande de metaheurísticas já muito bem testadas na literatura, dentre as quais podem ser citadas Algoritmos Genéticos (*Genetic Algorithms*), *Simulated Annealing*, *Greedy Randomized Adaptative Search Procedure*

(GRASP), Busca Tabu (*Tabu Search*), *Iterated Local Search* (ILS), Colônia de Formigas (*Ant Colony*), e *Variable Neighbourhood Search* (VNS).

Diversas metaheurísticas, como ILS, já definem em seu *framework* a existência de procedimentos de *busca local*. Outras metaheurísticas, como algoritmos genéticos, não obrigam a sua utilização, mas podem ter sua performance melhorada ao utilizar algum procedimento de busca local. Estes procedimentos começam com uma solução inicial e iterativamente tentam substituir a solução corrente por uma melhor, explorando a vizinhança com respeito a solução corrente, de modo que se alcance um ótimo local (Blum & Roli, 2003).

1.6 Metodologia

1.6.1 Modelo analítico

Com este trabalho pretende-se verificar a viabilidade de aplicação de heurísticas em dois problemas de otimização combinatória que envolvem loteamento: $1|ST_{sd,b}|\sum T_j$ e $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$.

Primeiro, deve-se identificar os métodos de resolução já disponíveis na literatura para cada um dos problemas. Para o problema $1|ST_{sd,b}|\sum T_j$ já se encontra na literatura a implementação de um algoritmo genético. Para o problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, no melhor do nosso conhecimento, não há implementação de metaheurística disponível na literatura, somente uma implementação de um algoritmo *Branch and Bound*.

Num momento inicial, deve ser feito um estudo aprofundado destes algoritmos para posterior implementação dos mesmos, com objetivo de comparação. Para o problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, como não há implementação disponível de metaheurística, deve ser feito um estudo aprofundado das características deste problema, como representação de uma solução que possa ser eficientemente manipulada por uma metaheurística e procedimentos de busca local. Deste modo, torna-se necessário buscar na literatura implementações para problemas semelhantes que possam ser adaptadas. Um passo seguinte é o desenvolvimento de um modelo matemático já que, no melhor do nosso conhecimento, não há um modelo específico para este problema.

Após a implementação de cada um destes métodos disponíveis na literatura, o passo seguinte é a proposição de metaheurísticas específicas, considerando as particularidades de cada problema.

Um conjunto específico de problemas teste (instâncias) será avaliado por cada uma das abordagens, e o valor da função objetivo e o tempo computacional serão utilizados como métricas a serem comparada.

A fim de garantir a consistência dos resultados, experimentos serão realizados de forma a comparar as implementações disponíveis na literatura e as que serão desenvolvidas neste trabalho. Pretende-se utilizar a metodologia DOE (Desenho de Experimentos) no desenvolvimento dos experimentos.

Após a conclusão destes passos espera-se que o objetivo, criar heurísticas que melhorem os resultados de algoritmos da literatura, para cada um dos problemas, seja alcançado.

1.6.2 Fonte de dados

Os dados serão obtidos a partir de fontes disponíveis na literatura. Um algoritmo exato (*Branch and Bound*), proposto no trabalho de Mazdeha et al. (2012) para o problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, será usado para comparação. Um algoritmo exato (Modelo de Programação Inteira) e uma heurística (algoritmo genético) propostos por Chantaravarapan et al. (2003) para o problema $1|ST_{sd,b}|\sum T_j$ serão usados para comparação. Os dados experimentais serão retirados destes trabalhos.

Parte I

O problema $1|ST_{sd,b}|\Sigma T_j$

Capítulo 2

Problema de sequenciamento em uma máquina com tempos de preparação dependentes da sequência das famílias das tarefas

Neste capítulo, será tratado especificamente de um problema da programação da produção que envolve o sequenciamento de tarefas em uma máquina e considerações sobre o tempo de preparação dependente da sequência das famílias das tarefas com o objetivo de encontrar um sequenciamento com o menor atraso total. Primeiramente, na Seção 2.1, é feita uma introdução sobre o problema, seguido de uma revisão de literatura na Seção 2.2. As heurísticas propostas são explicadas na Seção 2.3. Os experimentos computacionais e os resultados alcançados estão organizado na Seção 2.4. Finalmente, na Seção 2.5, estão as conclusões e trabalhos futuros para este primeiro problema abordado.

2.1 Introdução

A programação ou sequenciamento da produção é um processo de tomadas de decisão que ocorre em sistemas de manufaturas. Problemas de programação da produção têm sido investigados desde meados dos anos 50 (Allahverdi et al., 2008) e ainda na atualidade continuam sendo muito estudados. A importância de se estudar tais problemas provém, principalmente, por dois aspectos: o primeiro diz respeito a sua importância prática, com várias aplicações em diversos setores, como indústrias

química, metalúrgica e têxteis. O segundo aspecto é sobre a dificuldade para se resolver a maioria dos problemas de sequenciamento, pois eles pertencem a classe $\mathcal{NP}$-Difícil de problemas.

O problema da programação focado nesta primeira parte do trabalho é descrito a seguir. Há um conjunto de n tarefas a serem processadas em uma máquina sem interrupção ou preempção. As tarefas são divididas em F famílias e estão disponíveis num tempo zero. Cada família l tem n_l tarefas, tal que $n_1+n_2+\ldots+n_l+\ldots+n_F = n$. $f(j)$ denota a família da tarefa j. O tempo de processamento e a data de entrega da tarefa j são conhecidos e definidos por p_j e d_j, respectivamente. Há um tempo de preparação s_{jk} entre as tarefas j e k se a tarefa k é processada imediatamente depois da tarefa j. Se as tarefas j e k são da mesma família, $f(j) = f(k)$, não há nenhum tempo de preparação ($s_{jk} = 0$). O tempo de preparação é necessário, por exemplo, para o ajuste de ferramentas, limpeza, posicionamento de acessórios, inspeção de materiais, dentre outros. O tempo de preparação é dependente da sequência, isto é, $s_{jk} \neq s_{kj}$. O problema consiste em encontrar a ordem de processamento das tarefas (sequência) de modo a minimizar o atraso total das tarefas com relação a suas datas de entregas.

O tempo de conclusão de uma tarefa j é definido como C_j. Se uma tarefa termina antes da sua data de entrega, não há atraso ($T_j = 0$). Caso contrário, o atraso incorrido é dado por $C_j - d_j$. Logo, $T_j = max(C_j - d_j, 0)$. O atraso total das tarefas (função objetivo) é computado como:

$$f = \sum_{j=1}^{n} T_j \tag{2.1}$$

O problema é denotado por $1|ST_{sd,b}|\sum T_j$ seguindo a notação de três campos $\alpha|\beta|\gamma$ introduzida por Graham et al. (1979) e adaptada por Allahverdi et al. (2008), onde 1 denota o ambiente com uma única máquina, $ST_{sd,b}$ é a informação do tempo de preparação dependente da sequência dos lotes ou famílias e $\sum T_j$ é o critério do atraso total.

Uma vez que o problema de minimização de atraso total em uma máquina sem famílias é $\mathcal{NP}$-Difícil (Du & Leung, 1990), segue que o problema $1|ST_{sd,b}|\sum T_j$ é pelo menos $\mathcal{NP}$-Difícil no senso comum.

O critério de atraso total é muito importante em sistemas de manufaturas, porque podem existir diversos custos quando uma tarefa é entregue com atraso. Dentre estes podem ser citados: multas contratuais, perda de credibilidade resultando em alta probabilidade de se perder um cliente para alguma ou para todas as tarefas futuras, e danos na reputação da empresa que pode afastar outros clientes.

O restante deste capítulo está organizado como segue. Na Seção 2.2 é feita uma revisão de literatura de trabalhos relacionados ao problema $1|ST_{sd,b}|\sum T_j$. Na Seção 2.3 são descritos os métodos usados nas heurísticas ILS propostas. Os experimentos computacionais são apresentados na Seção 2.4. Finalmente, na Seção 2.5, estão as conclusões do trabalho para o problema $1|ST_{sd,b}|\sum T_j$.

2.2 Revisão de literatura

Problemas de programação da produção que consideram explicitamente tempos de preparação são de grande importância em sistemas de manufatura. Uma extensa revisão de literatura para estes problemas, considerando diferentes ambientes de produção, foi realizado por Allahverdi et al. (2008).

O problema de sequenciamento em uma máquina quando se considera tempos de preparação dependente da sequência das tarefas e minimização do atraso total, $1|ST_{sd}|\sum T_j$, tem sido abordado por vários trabalhos na literatura.

Uma das primeiras referências remete a Ragatz (1993), onde é proposto um algoritmo *Branch and Bound* (B&B) avaliado para instâncias com até 16 tarefas. Rubin & Ragatz (1995) aplicaram um Algoritmo Genético (AG) e um algoritmo de busca aleatória (*Random Search* - RS) a um conjunto de instâncias similares àquelas de Ragatz (1993) e cujos resultados são comparados à implementação do B&B. O AG e RS foram testados em problemas com até 45 tarefas, mostrando que os resultados do AG e RS são competitivos aos alcançados pelo B&B e que, em termos de qualidade de soluções e tempo de execução, em muitos casos, a implementação RS é melhor que o AG. Esta conclusão leva Tan & Narasimhan (1997) a proporem uma implementação de *Simulated Annealing* (SA) que é testada para um conjunto de problemas testes com até 50 tarefas e cujos resultados são comparados à RS. Os resultados do experimento demostram superioridade da implementação SA. Duas outras implementações de AG também são desenvolvidas nos trabalhos de Armentano & Mazzini (2000) e França et al. (2001), sendo que este último apresenta resultados para problemas com até 100 tarefas.

A pesquisa realizada em Tan et al. (2000) é uma extensão dos trabalhos de Rubin & Ragatz (1995) e Tan & Narasimhan (1997). Neste trabalho são comparados quatro implementações: o B&B de Ragatz (1993), o AG e RS de Rubin & Ragatz (1995) e a implementação de SA de Tan & Narasimhan (1997). A conclusão apresentada é que SA e RS possuem performance melhor. Gagné et al. (2002), por sua vez, propõe uma implementação da metaheurística *Ant Colony Optimization*

(ACO) que se mostra competitiva aos melhores resultados de Tan et al. (2000) e melhor em termos de tempo computacional.

Gupta & Smith (2006) apresentaram duas heurísticas: uma heurística *Problem Space-based Local Search* (PSLS) e uma heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) hibridizado com *Path Relinking* (PR). O autor conclui que a heurística PSLS possui performance semelhante quando comparada a heurística ACO de Gagné et al. (2002) e requer menor tempo computacional. A heurística GRASP, no entanto, requer um tempo computacional maior que a heurística ACO, porém gera soluções de melhor qualidade.

Mais recentemente, outras heurísticas foram apresentadas. Uma heurística ACO foi proposta por Liao & Juan (2007), e que considera também a versão ponderada do mesmo ($1|ST_{sd}|\sum w_jT_j$). Uma heurística *Iterated Local Search* (ILS) fora proposta por Arroyo et al. (2009), e que utiliza a heurística GRASP para gerar a solução inicial. Os resultados são comparados com o algoritmo GRASP de Gupta & Smith (2006) e com a heurística ACO de Liao & Juan (2007), fornecendo resultados competitivos com os da literatura. Ying et al. (2009) propõe a implementação de uma heurística *Iterated Greedy*. O autor trabalha também a versão ponderada e os resultados indicam que sua implementação é altamente competitiva com aquelas do estado da arte para este problema, conseguindo melhorar o custo de diversas soluções. Diversas outras implementações foram experimentadas como uma heurística GRASP (Akrout et al., 2012), um algoritmo B&B (Sewell et al., 2012) e um AG (Sioud et al., 2012). Porém, os resultados mais representativos, com respeito a encontrar as soluções ótimas, foram alcançados recentemente por Tanaka & Araki (2012) onde todas as instâncias do *benchmark* de Rubin & Ragatz (1995) foram resolvidas de forma ótima e quase todas as do *benchmark* de Gagné et al. (2002) também o foram. O autor utiliza um algoritmo exato baseado no método SSDP (*Successive Sublimation Dynamic Programming*). O contraponto é que o método requer muito tempo computacional.

Alguns estudos foram realizados sobre o problema de sequenciamento em uma máquina com tarefas agrupadas em famílias e tempos de preparação independente das famílias ($ST_{si,b}$). Nos trabalhos realizados por Kacem (2006) e Schaller (2007) foram propostos algorítimos B&B e heurísticas para o problema $1|ST_{si,b}|\sum T_j$. Para a minimização de adiantamentos e atrasos ($\sum E_j + \sum T_j$), métodos exatos e heurísticos foram desenvolvidos por Schaller & Gupta (2008). Abordagens exatas também foram usadas por Pan & Su (1997) e Baker & Magazine (2000) para resolver o problema $1|ST_{si,b}|L_{max}$, onde L_{max} é o *lateness* máximo.

No entanto, há poucos artigos para o problema de programação da produção

em uma máquina com tempos de preparação dependentes da sequência e das famílias ($1|ST_{sd,b}|*$). Van der Veen et al. (1998) abordou o problema $1|ST_{sd,b}|C_{max}$ para minimizar o tempo de conclusão máximo (C_{max}). Para o problema de minimização do tempo total de conclusão ($1|ST_{sd,b}|\sum C_j$), Karabati & Akkan (2005) propuseram um algoritmo B&B. No artigo de Jin et al. (2010) é abordado o problema $1|ST_{sd,b}|L_{max}$, no qual os autores usam uma heurística de busca tabu.

Chantaravarapan et al. (2003) abordaram o mesmo problema tratado nesta parte do trabalho. Os autores propuseram um algoritmo genético hibrido (HGA). O HGA foi mais eficiente em minimizar o atraso total quando comparado com outros algoritmos heurísticos.

2.3 Heurísticas ILS propostas

Neste capítulo, são desenvolvidos três algoritmos heurísticos baseados na metaheurística *Iterated Local Search* (ILS) para obter soluções aproximadas de alta qualidade para o problema $1|ST_{sd,b}|\sum T_j$. A ILS (Lourenço et al., 2003) é uma metaheurística simples, robusta e efetiva que aplica repetidamente uma busca local a soluções obtidas ao perturbar soluções ótimas locais previamente visitadas. Primeiramente é apresentada uma heurística com os componentes básicos de ILS. A seguir, a heurística é melhorada com a adoção de um controle automático da perturbação. Também é avaliada uma heurística baseada em ILS com um procedimento de intensificação baseado em *Path Relinking* e conjunto de soluções elite.

Quatro métodos são necessários para implementar um algoritmo ILS básico: um método "CONSTRUCTION", que gera uma solução inicial S viável, um método "PERTURBATION" que perturba a solução corrente S levando a alguma solução intermediária S_1, um procedimento de busca local "LOCAL_SEARCH" que melhora S_1 obtendo um ótimo local S_2, e um critério de aceitação ("ACCEPTANCE_CRITERION") que decide em qual solução a próxima perturbação será aplicada. O critério de aceitação define um compromisso entre diversificação e intensificação. Uma intensificação muito forte é alcançada se apenas soluções melhores que a corrente são aceitas (S_2 melhor que S). No outro extremo, há grande diversificação quando soluções são aceitas independente do valor da função objetivo. ILS é uma poderosa técnica de otimização, e tem sido aplicada para uma variedade de problemas de otimização combinatória, tais como o problema do caixeiro viajante (Martin & Otto, 1996), programação de tarefas no ambiente *job shop* (Lourenço, 1995) e no ambiente *flowshop* (Dong et al., 2009).

A primeira heurística implementa os componentes básicos de uma heurística ILS. Na segunda heurística proposta é utilizado controle automático da perturbação cuja intensidade é aumentada à medida que o procedimento de busca local não consegue gerar novas soluções melhores. Na terceira versão da heurística ILS, além do controle automático da perturbação, propõe-se a utilização de uma estratégia de intensificação baseada na técnica de *Path Relinking* (PR). Nesta versão da heurística, há também um conjunto de soluções elite, que armazena soluções diferentes de alta qualidade, que é construído e atualizado com as novas soluções geradas durante as iterações da heurística. O procedimento PR gera novas soluções explorando trajetórias que conectam duas soluções de alta qualidade: a solução retornada pela busca local e uma solução retornada do conjunto de soluções elite.

Nas próximas subseções são descritos os métodos utilizados pelas heurísticas ILS propostas.

2.3.1 Representação de uma solução

Uma solução (sequenciamento) do problema $1|ST_{sd,b}|\sum T_j$ é representada por uma permutação de n tarefas $S = \{j_1, j_2,..., j_n\}$, onde j_k é a k-ésima tarefa na sequência de processamento. Por exemplo, considere a instância com $n = 7$ tarefas que é especificada na Tabela 2.1. A família, o tempo de processamento e a data de entrega de cada tarefa estão na Tabela 2.1(a), e os tempos de preparação entre as famílias são mostrados na Tabela 2.1(b). Para o sequenciamento $S = \{7, 1, 5, 4, 2, 6, 3\}$ os tempos de conclusão e atrasos das tarefas são ($C_7 = 2$, $C_1 = 5$, $C_5 = 10$, $C_4 = 12$, $C_2 = 14$, $C_6 = 19$, $C_3 = 23$) e ($T_7 = 0$, $T_1 = 3$, $T_5 = 2$,$T_4 = 1$, $T_2 = 7$, $T_6 = 4$, $T_3 = 5$), respectivamente. Logo, o atraso total é 22. Este sequenciamento é ilustrado na Figura 2.1. Note que há a formação de quatro lotes e três tempos de preparação são requeridos (tempo de preparação não é considerado no início da sequência).

Tabela 2.1. Dados de entrada para uma instância.

Tarefa i	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$f(i)$	1	2	1	2	2	1	2
d_i	2	7	18	11	8	15	3
p_i	1	2	4	2	4	3	2

(a) Família, data de entrega e tempo de processamento

s_{ik}	**1**	**2**
1	0	1
2	2	0

(b) Tempo de preparação entre famílias.

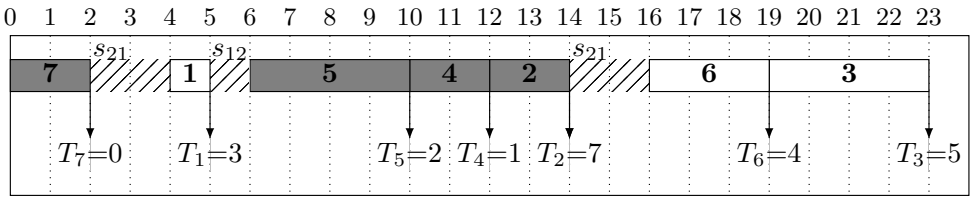


Figura 2.1. Exemplo de um sequenciamento $\mathbf{S} = \{\mathbf{7}, \mathbf{1}, \mathbf{5}, \mathbf{4}, \mathbf{2}, \mathbf{6}, \mathbf{3}\}$.

2.3.2 Construção da solução inicial

Para construir uma solução inicial para o ILS, foi utilizado um algoritmo baseado na heurística NEH (Nawaz et al., 1983) adaptada para a minimização do atraso total. Nesta heurística, primeiramente as tarefas são arranjadas em ordem não-decrescente das suas datas de entrega (regra *Earliest Due Date*) formando a lista ordenada $J = \{j_1, ...,j_n\}$ de tarefas. A primeira tarefa de J é inserida na sequência parcial S ($S = \{j_1\}$). Cada tarefa restante j_i de J (i = 2,...,n) é inserida em todas as posições possíveis da sequência S gerando i sequências parciais. Por exemplo, para $i = 2$ serão geradas duas sequências parciais: $\{j_2, j_1\}$ e $\{j_1, j_2\}$. Das i sequências, a melhor sequência parcial (em relação ao atraso total parcial) é atribuída a S. A heurística termina quando todas as tarefas da lista J são inseridas na sequência S.

2.3.3 Busca local

A busca local é usada para melhorar a solução inicial S e as soluções obtidas pelo procedimento de perturbação. Busca local é um método que começa com uma solução S, e gera uma vizinhança que contém soluções que são alcançadas através de movimentos individuais realizados na solução corrente. Desta vizinhança, uma solução que é melhor que a solução corrente é selecionada. A solução escolhida torna-se a nova solução corrente e o processo continua até que um ótimo local seja alcançado.

Neste trabalho foi utilizado o procedimento de busca local proposto por Ruiz & Stützle (2008) que é adaptado para sequenciamento em uma máquina e minimização do atraso total. O procedimento de busca local é baseado na vizinhança por inserção. Esta vizinhança consiste em remover cada tarefa j da sua posição original e inseri-la nas $n - 1$ posições restantes. Este movimento gera uma vizinhança de tamanho máximo de $(n - 1)^2$. A solução corrente é substituída por uma solução vizinha que

melhore o valor do atraso total. O procedimento é repetido enquanto há melhoria na solução corrente.

O procedimento de busca local utilizado neste trabalho é mostrado no Algoritmo 1. Inicialmente é inicializado o conjunto J que contém todas as tarefas (linha 5). A seguir sorteia-se uma tarefa j deste conjunto que é removida do conjunto J para que ela não possa ser sorteada novamente (linhas 7 e 8). Para reduzir o tamanho da vizinhança é utilizado o parâmetro γ que define a probabilidade para uma tarefa j ser avaliada em todas as posições da solução corrente. Se $\gamma = 1.0$, a vizinhança inteira é avaliada assim como em Ruiz & Stützle (2008). Observe que nas linhas 9 e 10 insere-se a tarefa j, com uma certa probabilidade γ, em cada posição da solução S. O algoritmo possui também o parâmetro FFI (*Flag First Improvement*) que define o critério de escolha da solução vizinha a ser explorada na próxima iteração. Note que se $FFI = 1$ (linha 14) na busca local é usada a estratégia de escolha do primeiro vizinho melhor (*First Improvement*). Caso contrário, o melhor da vizinhança (*Best Improvement*) é usado na próxima iteração da busca local.

Algoritmo 1: LOCAL_SEARCH (S, γ, FFI)

```
   output: Improved solution S
 1 begin
 2     improve := true;
 3     while improve do
 4         improve := false;
 5         J := {j1,...,jn} /*Set of n jobs*/;
 6         for i := 1 to n do
 7             j := randomly select a job from J;
 8             J := J − {j} ;
 9             if rand(0..1) ≤ γ then
10                 S' := best solution obtained by inserting j in all positions of
                   solution S;
11                 if f(S') < f(S) then
12                     S := S';
13                     improve := true;
14                     if FFI = 1 then
15                         Go to line 3 /*First Improvement*/
16     return S
```

2.3.4 Perturbação

O objetivo do método de perturbação em uma heurística ILS é escapar de uma solução ótima local. A perturbação de uma solução deve ser forte o bastante para

sair do mínimo local corrente e habilitar a busca local para encontrar um novo, possivelmente melhor, mínimo local. Ao mesmo tempo, a perturbação deve ser fraca o bastante para manter boas características do mínimo local corrente (Lourenço et al., 2010). Neste trabalho, foi utilizado um procedimento de perturbação similar ao proposto por Rego et al. (2012). Uma solução é perturbada realizando um número de inversões entre tarefas e é aplicado em níveis. Se a melhor solução não é melhorada pela busca local por um número de N iterações consecutivas, o nível da perturbação é incrementado, caso contrário o nível da perturbação retorna ao seu valor mais baixo.

O nível d da perturbação é um valor no intervalo $d \in [1, d_{max}]$, $d_{max} \leq (n/2-1)$. No nível d, $d+1$ inversões são feitas na solução. Desta maneira, no primeiro nível duas inversões são aplicadas, enquanto que no nível mais alto $n/2$ inversões são feitas.

Para perturbar uma solução S, uma posição j $(0 \leq j \leq n-2d-2)$ é aleatoriamente escolhida de S. As inversões são feitas no subconjunto de tarefas consecutivas de S nas posições $j, j+1,...,j+2d+1$. Então, as inversões são aplicadas entre os pares das posições: $(j, j+2d+1)$, $(j+1, j+2d)$,...,$(j+d, j+d+1)$. Desta maneira, $d+1$ movimentos de inversões são realizados em cada chamada do procedimento de perturbação.

2.3.5 Intensificação *Path Relinking*

A técnica de *Path Relinking* (PR) foi proposta originalmente por Glover (1996) como um mecanismo para combinar diversificação e intensificação explorando trajetórias que conectam soluções de alta qualidade (elite) previamente produzidas durante o processo de busca. PR precisa de um par de soluções, digamos S_i (solução inicial) e S_g (solução guia), $S_i \neq S_g$. Um caminho que conecta S_i a S_g é gerado aplicando-se movimentos de vizinhança na solução inicial, na qual progressivamente vão sendo inseridos atributos da solução guia (Laguna & Martí, 1999).

Neste trabalho, a variante *Mixed Path Relinking* (Resende & Ribeiro, 2010) é usada como uma estratégia de intensificação que tenta melhorar cada ótimo local obtido pelo procedimento de busca local. A solução inicial S_i é a solução retornada da busca local e a solução guia S_g é uma solução selecionada aleatoriamente de um conjunto de soluções elite, chamado de *EliteSet*. Este conjunto representa um *pool* das melhores soluções diferentes encontradas pelo algoritmo ILS. Dois caminhos são iniciados simultaneamente: um de S_i levando a S_g e outro levando de S_g até S_i. Estes dois caminhos encontram-se em alguma solução viável que conecta

S_i e S_g com um único caminho.

Em cada passo, movimentos de troca são realizados em S_i (S_g), incorporando atributos de S_g (S_i). Estes movimentos modificam progressivamente S_i (S_g) e a transformam em outra solução S_g (S_i). Novas soluções são analisadas e, se aplicável, a melhor solução é atualizada. O procedimento *Mixed PR* retorna a melhor solução encontrada durante a construção do caminho que conecta as soluções S_i e S_g.

Neste trabalho, a incorporação de atributos é realizada através da diminuição das diferenças entre S_i e S_g. O número de diferenças entre duas soluções é computada através da distância *Hamming*. Por exemplo, sejam os sequenciamentos $S_i = \{2, \mathbf{3}, \mathbf{1}, \mathbf{4}, \mathbf{5}, \mathbf{7}, \mathbf{6}\}$ e $S_g = \{2, 7, 5, 3, 4, 6, 1\}$. Este dois sequenciamentos possuem em comum somente a tarefa 2 na primeira posição. Nas demais posições de S_i, destacadas em negrito, as respectivas tarefas não coincidem com tarefas de S_g nas mesmas posições. Logo, o total de diferenças entre S_i e S_g, ou a distância *Hamming* D, entre os dois sequenciamentos é $D = 6$. Ao gerar os caminhos pelo procedimento *Mixed PR* a distância entre as duas soluções vai sendo diminuída gradativamente até que se encontrem em uma solução comum.

Um exemplo numérico de como o procedimento *Mixed PR* foi implementado neste trabalho é mostrado a seguir. Para o exemplo foi utilizada uma instância com 7 tarefas e 2 famílias, cujos dados são mostrados na Tabela 2.2. A Figura 2.2 mostra o exemplo gráfico das soluções geradas pelo procedimento.

Tabela 2.2. Instância utilizada no exemplo numérico da aplicação do *Mixed Path Relinking*.

Tarefa i	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$f(i)$	1	2	2	1	2	2	1
d_i	4	0	2	12	4	5	13
p_i	1	2	3	2	1	2	1

(a) Família, data de entrega e tempo de processamento.

s_{ik}	**1**	**2**
1	0	2
2	1	0

(b) Tempo de preparação entre famílias.

Como o procedimento constrói dois caminhos simultaneamente, a solução inicial e a solução guia são especificadas, respectivamente, por $S_i(k)$ e $S_g(k)$, sendo k a iteração (ou passo) considerado. No primeiro passo, $k = 0$, a solução inicial é a solução S_0, $S_i(0) = S_0$ e a solução guia é a solução S_{19}, $S_g(0) = S_{19}$. A distância entre as duas soluções é igual a $D = 6$. São geradas, então, 6 soluções $(S_1, S_2, \cdots, S_6)$ a

partir de $S_i(0)$ de modo que tenham uma menor distância com relação a $S_g(0)$. Por exemplo, a tarefa na segunda posição de $S_i(0)$ é a tarefa 3. Por outro lado, a tarefa na segunda posição de $S_g(0)$ é a tarefa 7. As tarefas 3 e 7, em $S_i(0)$, são trocadas, de forma que a tarefa 7 fique na segunda posição de $S_i(0)$, gerando a solução S_1 que possui distância *Hamming* $D = 5$. A melhor solução gerada no primeiro passo é escolhida para ser a nova solução guia do passo seguinte, $S_g(1) = S_2$ e a solução guia do passo anterior é escolhida para ser a solução inicial da próxima iteração, $S_i(1) = S_{19}$. O procedimento termina quando há uma solução que conecta S_i e S_g com um único caminho, no caso a solução S_{12}. A melhor solução encontrada é retornada pelo procedimento. Neste caso, a solução retornada é a solução S_{17} com valor da função objetivo igual a $f = 18$. Observe que o procedimento conseguiu encontrar uma solução melhor que as soluções de entrada S_0 e S_{19}.

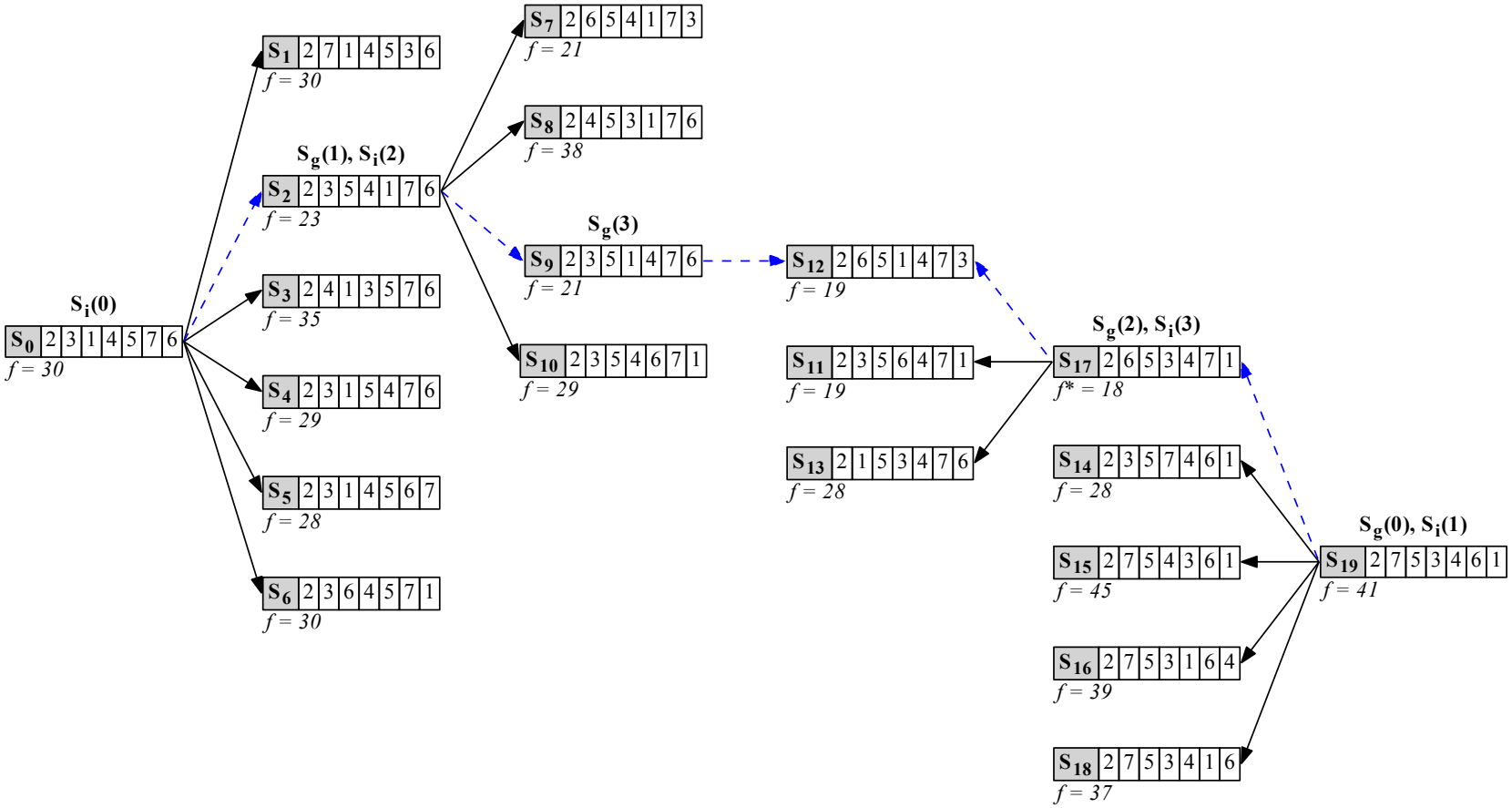


Figura 2.2. Exemplo gráfico do funcionamento do algoritmo *Mixed Path Relinking*.

2.3.6 Estrutura do algoritmo ILS básico

Foi feita a implementação de um algoritmo ILS sem controle automático da perturbação e sem a utilização de *Path Relinking*. O pseudocódigo deste algoritmo, chamado ILS_BASIC, é apresentado no Algoritmo 2. O algoritmo tem quatro parâmetros de entrada: o parâmetro d (que define o nível da perturbação), γ (a probabilidade de uma solução escolhida na busca local ter sua vizinhança avaliada), β (a probabilidade de aceitação da solução melhorada mesmo que seja pior que a solução corrente) e FFI (que define qual o critério de seleção do vizinho será utilizado na busca local).

Neste algoritmo, na linha 2 uma solução inicial é construída e é melhorada pelo procedimento de busca local (linha 3). As iterações do algoritmo ILS são computadas nas linhas 5-14, até que o critério de parada seja satisfeito. Durante cada iteração, a solução corrente S é perturbada (linha 6) e melhorada pela busca local, obtendo uma solução S_2 (linha 7). Nas linhas 8 e 9, se a solução S_2 é melhor que a melhor solução obtida até o momento, atualiza-se a melhor solução. Nas linhas 10-14 é testado o critério de aceitação. Se S_2 é melhor que a solução corrente S, então ela é aceita (S_2 substitui S). Caso contrário, ela pode ser aceita com uma pequena probabilidade β. A melhor solução encontrada durante todas as iterações é retornada pelo algoritmo (linha 15).

Algoritmo 2: ILS_BASIC (d, β, γ, FFI)

```
   output: Best Solution S*
 1 begin
 2 |  S := CONSTRUCTION();
 3 |  S := LOCAL_SEARCH(S, γ, FFI);
 4 |  S* = S;
 5 |  while not stopping criterion do
 6 |  |  S1 := PERTURBATION(S, d);
 7 |  |  S2 := LOCAL_SEARCH(S1, γ, FFI);
 8 |  |  if f(S2) < f(S*) then
 9 |  |  |  S* := S2;
10 |  |  if f(S2) ≤ f(S) then
11 |  |  |  S := S2;
12 |  |  else
13 |  |  |  if rand(0..1) ≤ β then
14 |  |  |  |  S := S2;
15 |  return S*
```

2.3.7 Estrutura do algoritmo ILS com controle dinâmico da perturbação

Uma descrição geral do pseudocódigo do algoritmo proposto com controle automático da perturbação (chamado ILS_DP) é apresentado no Algoritmo 3. O algoritmo tem cinco parâmetros de entrada: o parâmetro N (que controla a atualização do nível da perturbação), d_{max} (o nível máximo da perturbação), γ (probabilidade de uma solução escolhida na busca local ter sua vizinhança avaliada), β (probabilidade de aceitação da solução melhorada mesmo que seja pior que a solução corrente) e FFI (que define qual o critério de seleção do vizinho será utilizado na busca local)

Na linha 2 uma solução inicial é construída e melhorada pelo procedimento de busca local (linha 3). Na linha 4 o nível da perturbação é inicializado ($d = 1$). As iterações do algoritmo ILS são computadas nas linhas 7-22 até que o critério de parada seja satisfeito. Durante cada iteração, a solução corrente S é perturbada (linha 8) e melhorada pela busca local, obtendo uma solução S_2 (linha 9). Nas linhas 10-13, se a solução S_2 é melhor que a melhor solução obtida até o momento, o nível da perturbação é alterado para o seu menor valor ($d = 1$). Se a melhor solução não é melhorada durante N iterações consecutivas, o nível da perturbação é incrementado (veja linhas 15-17). O maior valor para o nível da perturbação é d_{max}. Nas linhas 18-22 é testado o critério de aceitação. Se S_2 é melhor que a solução corrente S então ela é aceita (S_2 substitui S), caso contrário ela pode ser aceita com uma pequena probabilidade β. A melhor solução encontrada durante todas as iterações é retornada pelo algoritmo (linha 23).

2.3.8 Estrutura do algoritmo ILS com *Path Relinking*

Uma descrição geral do pseudocódigo do algoritmo proposto ILS com PR (chamado ILS_DP+PR) é apresentado no Algoritmo 4. O algoritmo possui seis parâmetros de entradas: N (parâmetro que controla a atualização do nível da perturbação), $sizeElite$ (o tamanho máximo do conjunto elite), d_{max} (o nível máximo da perturbação), γ (a probabilidade de uma solução escolhida na busca local ter sua vizinhança avaliada), β (a probabilidade de aceitação da solução melhorada mesmo que seja pior que a solução corrente) e FFI (que define qual o critério de seleção do vizinho será utilizado na busca local).

Na linha 2 uma solução inicial é construída e é melhorada pelo procedimento de busca local (linha 3). O conjunto elite $EliteSet$ é inicializado com a primeira solução melhorada S (linha 4). Na linha 5 o nível da perturbação é inicializado ($d = 1$).

Algoritmo 3: ILS_DP (N, d_{max}, β, γ, FFI)

```
   output: Best Solution S*
 1 begin
 2 |  S := CONSTRUCTION();
 3 |  S := LOCAL_SEARCH(S, γ, FFI);
 4 |  d := 1;
 5 |  cont := 0;
 6 |  S* := S; //best solution
 7 |  while not stopping criterion do
 8 |  |  S1 := PERTURBATION(S, d);
 9 |  |  S2 := LOCAL_SEARCH(S1, γ, FFI);
10 |  |  if f(S2) < f(S*) then
11 |  |  |  S* := S2;
12 |  |  |  cont := 0;
13 |  |  |  d := 1;
14 |  |  else
15 |  |  |  cont := cont + 1;
16 |  |  |  if cont mod N = 0 then
17 |  |  |  |  d := min(d + 1, dmax);
18 |  |  if f(S2) ≤ f(S) then
19 |  |  |  S := S2;
20 |  |  else
21 |  |  |  if rand(0..1) ≤ β then
22 |  |  |  |  S := S2;
23 |  return S*
```

As iterações do algoritmo ILS são computadas nas linhas 8 a 26 até que o critério de parada seja satisfeito. Durante cada iteração, a solução corrente S é perturbada (linha 9) e melhorada pelo procedimento LOCAL_SEARCH, obtendo uma solução S_2 (linha 10). Na linha 12, *MIXED_PR* é aplicado entre S_2 e uma solução S_3 selecionada aleatoriamente do conjunto *EliteSet* (linha 11). *MIXED_PR* retorna a melhor solução encontrada S_4. O conjunto elite é atualizado com as soluções S_2 e S_4. Este conjunto armazena no máximo *sizeElite* soluções de alta qualidade. Quando o conjunto elite está completo, uma solução X é adicionada a *EliteSet* se $X \notin EliteSet$ e é melhor que a pior solução em *EliteSet* (neste caso X é adicionada em *EliteSet* no lugar da pior solução).

Nas linhas 14 a 17, se a solução S_4 é melhor que a melhor solução obtida até o momento, o nível da perturbação é atribuído ao seu menor valor ($d = 1$). Se a melhor solução não é melhorada durante N iterações consecutivas, o nível da perturbação é incrementado (veja linhas 20 e 21). O maior valor do nível da perturbação é d_{max}. Nas linhas 22 a 26 é testado o critério de aceitação. Se S_4 é melhor que a solução

corrente S então ela é aceita (S_4 substitui S). Caso contrário, ela pode ser aceita com uma pequena probabilidade β. A melhor solução do conjunto elite é retornada pelo algoritmo (linha 27).

Algoritmo 4: ILS_DP+PR (N, $sizeElite$, d_{max}, β, γ, FFI)

```
   output: Best solution of EliteSet
 1 begin
 2     S := CONSTRUCTION();
 3     S := LOCAL_SEARCH(S, γ, FFI);
 4     EliteSet := {S};
 5     d := 1;
 6     cont := 0;
 7     f(S*) := f(S); //the best value of total tardiness
 8     while not stopping criterion do
 9         S1 := PERTURBATION(S, d);
10         S2 := LOCAL_SEARCH(S1, γ, FFI);
11         S3 := Randomly select a solution from EliteSet;
12         S4 := MIXED_PR(S2, S3);
13         EliteSet := UPDATE_ELITE_SET(S2, S4);
14         if f(S4) < f(S*) then
15             f(S*) := f(S4);
16             cont := 0;
17             d := 1;
18         else
19             cont := cont + 1;
20             if cont mod N = 0 then
21                 d := min(d + 1, dmax);
22         if f(S4) ≤ f(S) then
23             S := S4;
24         else
25             if rand(0..1) ≤ β then
26                 S := S4;
27     return Best solution of EliteSet
```

2.4 Análise de desempenho computacional e estatística

Neste trabalho, foram comparadas as implementações das heurísticas ILS propostas com um Algoritmo Genético Híbrido (HGA) proposto por Chantaravarapan et al. (2003) para resolver o problema $1|ST_{sd,b}|\sum T_j$. O algoritmo HGA foi reimplementado seguindo o artigo original (Chantaravarapan et al., 2003). Também são

analisados a eficiência do controle automático da perturbação usado no algoritmo ILS e da inclusão do procedimento de *Path Relinking*; consequentemente foram comparadas as versões ILS_DP e ILS_DP+PR com a versão básica que usa um nível de perturbação fixo (chamado ILS_BASIC). Todos os algoritmos foram codificados em C++ e executados em uma máquina com processador Intel(R) Xeon(R) CPU X5650 @ 2.67GHz e 48 GB de RAM.

2.4.1 Geração de problemas teste

As instâncias do problema $1|ST_{sd,b}|\sum T_j$ foram geradas de acordo com o trabalho de Jin et al. (2010). Foram considerados problemas testes com o número de tarefas $n \in \{60,80,100\}$ e número de famílias $F \in \{2,3,4,5\}$. O tempo de processamento das tarefas (p_j) foram gerados aleatoriamente, com distribuição uniforme, sobre o intervalo $[1; 99]$. Os tempos de preparação são uniformemente distribuídos em três classes de intervalos: **classe S** em $[11; 20]$; **classe M** em $[51; 100]$ e **classe L** em $[101; 200]$. As datas de entrega são números inteiros no intervalo $[0; r\sum_{j=1}^{n} p_j]$, onde r é um parâmetro usado para controlar a amplitude da data de entrega, $r \in \{0{,}5; 1{,}5; 2{,}5; 3{,}5\}$. Combinando os parâmetros n, F, r e classes de tempo de preparação, tem-se 144 configurações possíveis. Para cada configuração, 10 instâncias foram geradas. Portanto, um total de 1440 instâncias foram testadas.

2.4.2 Métrica para avaliação dos algoritmos

A métrica comumente utilizada para avaliar a qualidade das soluções obtidas por algoritmos heurísticos é a medida do Desvio Percentual Relativo (RPD em inglês *Relative Percentage Deviation*) (Vallada et al., 2008):

$$RPD = \frac{f_{method} - f_{best}}{f_{best}} \times 100\% \tag{2.2}$$

onde f_{method} é a média das soluções (valor da função objetivo) obtido por um dado método (algoritmo) e f_{best} é a melhor solução obtida entre todos os algoritmos comparados.

A utilização da métrica RPD em problemas que consideram a minimização de atraso pode fazer com que ocorra divisão por zero (quando o atraso é zero para a melhor solução). Então, neste trabalho foi utilizado a medida do Índice de Desvio Relativo (RDI em inglês *Relative Deviation Index*) (Vallada et al., 2008) que

é computada da seguinte maneira:

$$RDI = \frac{f_{method} - f_{best}}{f_{worst} - f_{best}} \times 100\% \tag{2.3}$$

onde f_{worst} é a pior solução obtida entre todos os algoritmos comparados. Com esta medida, um índice entre 0 e 100 é obtido para cada método tal que uma boa solução terá um índice muito próximo de 0. Se todos os métodos encontram a mesma solução para uma determinada instância, o RDI será 0 para todos os métodos.

2.4.3 Calibração dos parâmetros

Para determinar a melhor configuração dos valores para os parâmetros de cada algoritmo, um experimento computacional foi realizado baseado na metodologia Desenho de Experimentos (DOE) (Montgomery, 2006) onde cada fator é um parâmetro controlado. O desenho fatorial completo é usado para todos os fatores de cada algoritmo.

Os experimentos foram realizados utilizando uma amostra de 100 instâncias diferentes do problema. Para cada instância os algoritmos foram executados cinco vezes utilizando cada configuração dos parâmetros, e o valor da função objetivo é utilizado para calcular o RPD (Equação (2.2)) em cada uma das cinco execuções. A utilização do RPD se justifica por não se verificar experimentalmente, nestas 100 instâncias, o atraso total zero em qualquer rodada. Então, esta medida é considerada como a variável de resposta nos experimentos estatísticos. Os resultados foram determinados através da Análise de Variância (ANOVA) paramétrica e testes de Comparações Múltiplas. Na realização da ANOVA, a normalidade foi verificada pelo teste de *Shapiro-Wilk W* e a homoscedasticidade pelo teste de *Levene*.

2.4.3.1 Calibração dos Parâmetros do ILS_BASIC

Para o algoritmo ILS_BASIC foram analisados quatro parâmetros: γ, β, FFI e d. O parâmetro d é um nível de perturbação fixo, utilizado somente no ILS_BASIC. Os seguintes conjuntos de valores foram testados: $\gamma \in \{0{,}1; 0{,}3; 0{,}6; 1{,}0\}$, $\beta \in \{0; 0{,}3; 0{,}6; 1{,}0\}$, $FFI \in \{0; 1\}$ e $d \in \{\lceil n/100 \rceil; \lceil n/20 \rceil; \lceil n/10 \rceil; \lceil n/7 \rceil; \lceil n/5 \rceil\}$. Note que ao todo foram testadas 160 configurações de parâmetros. Foram verificadas as três principais suposições da ANOVA paramétrica, para que os resultados fornecidos pelo teste tenham validade: normalidade, igualdade de variância (ou homoscedasticidade) e independência dos resíduos (ver Apêndice B.1).

A Tabela 2.3 mostra o resultado da ANOVA referente a calibração do ILS_BASIC. O *teste-F* na tabela ANOVA irá testar se há qualquer diferença significante entre as médias do RPD (variável de resposta) encontradas por cada uma das 160 configurações (tratamentos) utilizadas na calibração do ILS_BASIC. Esta tabela decompõe variação total entre todas as observações, na variação devido ao efeito dos tratamentos e na variação devida ao acaso ou resíduos (veja coluna "Fonte da Variação"). O número de graus de liberdade (veja coluna "'Graus de Liberdade") dos tratamentos é igual a $I - 1$, onde I é o número de tratamentos utilizados. No caso, $I = 160$ configurações, sendo o número de graus de liberdade associado aos tratamentos igual a 159. Já o número de graus de liberdade associados aos resíduos é igual a $I(J-1)$, onde J é o número de repetições. No caso, $J = 5$ repetições o que dá um total de 640 graus de liberdade. A coluna "Quadrados Médios" corresponde ao quociente entre a soma de quadrados pelo grau de liberdade da respectiva fonte de variação. O *valor de F*, que neste caso é igual a 723,85, é o quociente entre o quadrado médio dos tratamentos pelo quadrado médio do resíduos. Uma vez que o *valor-P* do teste-F, que é o valor de interesse, é menor que 0,05, há uma diferença estatisticamente significante de um dos conjuntos de parâmetros testados para o ILS_BASIC para outro conjunto, com um nível de confiança de 95%.

Tabela 2.3. Tabela ANOVA referente a calibração de parâmetros do algoritmo ILS_BASIC.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	55480,90	159	348,94	**723,85**	**0**
Resíduos	308,52	640	0,48		
Total	55789,40	799			

A ANOVA mostrou que há diferença significativa entre as configurações, mas não especifica qual configuração é a melhor ou não especifica onde estão os contrastes entre os tratamentos. Então, um teste de Comparações Múltiplas foi realizado para este fim. A Figura 2.3 mostra o gráfico de médias resultante do teste *Tukey* da Diferença Honestamente Significativa HSD (*Honestly Significant Difference*) com nível de confiança de 95% para as configurações testadas no algoritmo ILS_BASIC. Das 160 configurações testadas, por questão de facilidade de visualização, somente as configurações de 99 a 145 são mostradas no gráfico. Nesta figura é possível ver que existe diferença significativa entre as configurações, pois há diversos intervalos que não se sobrepõe, embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. Por exemplo, há diferença significativa entre as configurações 99 e 100 (primeira e segunda configuração mostradas no gráfico)

porque os respectivos intervalos não se sobrepõe. Para o algoritmo ILS_BASIC a configuração 143 (terceira da direita para esquerda) apresentou a melhor média e corresponde aos valores $\gamma = 1{,}0$; $\beta = 0{,}3$; $FFI = 1$ e $d = \lceil n/5 \rceil$. Como o valor encontrado para o parâmetro $d = \lceil n/5 \rceil$ é o maior dentre aqueles testados mais uma etapa de calibração foi realizada, visto que o maior valor possível da perturbação é $\lceil n/2 \rceil - 1$. Assim, mais três níveis para o parâmetro d foram testados: $d \in \{\lceil n/4 \rceil; \lceil n/3 \rceil; \lceil n/2 \rceil - 1\}$ sendo que aquele que na média obteve os melhores resultados foi $d = \lceil n/3 \rceil$.

Após as duas etapas de calibração os valores dos parâmetros utilizados nos experimentos finais para o algoritmo ILS_BASIC foram: $\gamma = 1{,}0$; $\beta = 0{,}3$; $FFI = 1$ e $d = \lceil n/3 \rceil$.

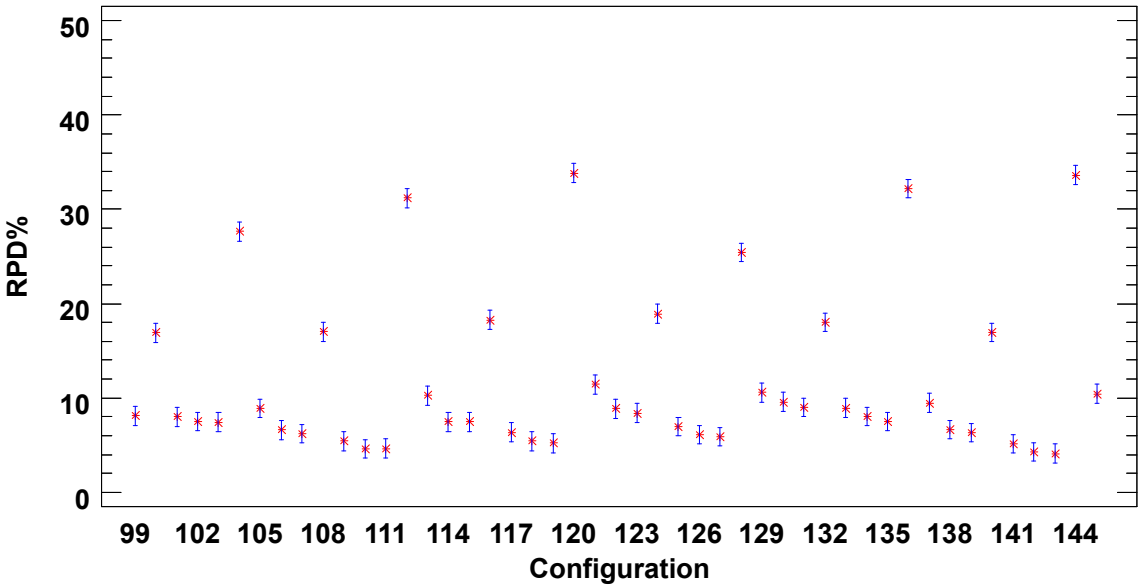


Figura 2.3. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 160 configurações de ILS_BASIC. Etapa 1 da calibração.

2.4.3.2 Calibração dos Parâmetros do ILS_DP

Para o algoritmo ILS_DP, cinco parâmetros foram ajustados: N, d_{max}, γ, β e FFI. Foram analisados os seguintes valores: $N \in \{1; 3; 5; 10\}$, $d_{max} \in \{\lceil n/10 \rceil; \lceil n/5 \rceil\}$, $\gamma \in \{0{,}1; 0{,}3; 0{,}6; 1{,}0\}$, $\beta \in \{0; 0{,}3; 0{,}6; 1{,}0\}$ e $FFI \in \{0; 1\}$. Note que ao todo foram testadas 256 configurações de parâmetros.

Os resultados foram analisadas através da ANOVA, seguindo o mesmo procedimento utilizado na calibração do ILS_BASIC (conforme explicado na Seção 2.4.3.1). A Tabela 2.4 mostra o resultado do teste ANOVA, com nível de confi-

ança de 95%. Como o *valor-p* é menor que 0,05 há diferença estatística entre as configurações testadas para o ILS_DP.

Tabela 2.4. Tabela ANOVA referente a calibração de parâmetros do algoritmo ILS_DP.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	23966,80	255	93,99	**672,02**	**0**
Resíduos	143,21	1024	0,14		
Total	24110,00	1279			

A Figura 2.4 mostra o gráfico de médias resultante do teste *Tukey* da Diferença Honestamente Significativa HSD (*Honestly Significant Difference*) com nível de confiança de 95% para as configurações testadas no algoritmo ILS_DP. Das 256 configurações testadas, por questão de facilidade de visualização, somente as configurações de 138 a 153 são mostradas no gráfico. Nesta figura é possível ver que existe diferença significativa entre as configurações pois há diversos intervalos que não se sobrepõe, mesmo embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. Para o algoritmo ILS_DP a configuração 142 apresentou a melhor média e corresponde aos valores $N = 1$; $d_{max} = \lceil n/5 \rceil$; $\gamma = 0{,}6$; $\beta = 0{,}6$ e $FFI = 1$. Como o valor encontrado para o parâmetro $d_{max} = \lceil n/5 \rceil$ é um valor limite mais uma etapa de calibração foi realizada, visto que o maior valor possível de perturbação é $\lceil n/2 \rceil - 1$. Assim, mais três níveis para d_{max} foram testados: $d_{max} \in \{\lceil n/4 \rceil; \lceil n/3 \rceil; \lceil n/2 \rceil - 1\}$ sendo que aquele que na média obteve os melhores resultados foi $d_{max} = \lceil n/3 \rceil$. Após as duas etapas de calibração os valores dos parâmetros utilizados nos experimentos finais para os algoritmos ILS_DP foram $N = 1$; $d_{max} = \lceil n/3 \rceil$; $\gamma = 0{,}6$; $\beta = 0{,}6$ e $FFI = 1$. O Apêndice B.2 contém mais detalhes dos testes estatísticos realizados.

2.4.3.3 Calibração dos Parâmetros do ILS_DP+PR

Para o algoritmo ILS_DP+PR, seis parâmetros foram ajustados: N, $sizeElite$, d_{max}, γ, β e FFI. Foram analisados os seguintes valores: $N \in \{1; 3; 5; 10\}$, $sizeElite \in \{10; 20\}$, $d_{max} \in \{\lceil n/10 \rceil; \lceil n/5 \rceil\}$, $\gamma \in \{0{,}1; 0{,}3; 0{,}6; 1{,}0\}$, $\beta \in \{0; 0{,}3; 0{,}6; 1{,}0\}$ e $FFI \in \{0; 1\}$. Note que ao todo foram testadas 256 configurações de parâmetros.

Os resultados foram analisadas através da ANOVA, seguindo o mesmo procedimento utilizado na calibração dos algoritmos ILS_BASIC e ILS_DP (conforme explicado nas Seções 2.4.3.1 e 2.4.3.2). A Tabela 2.5 mostra o resultado do teste

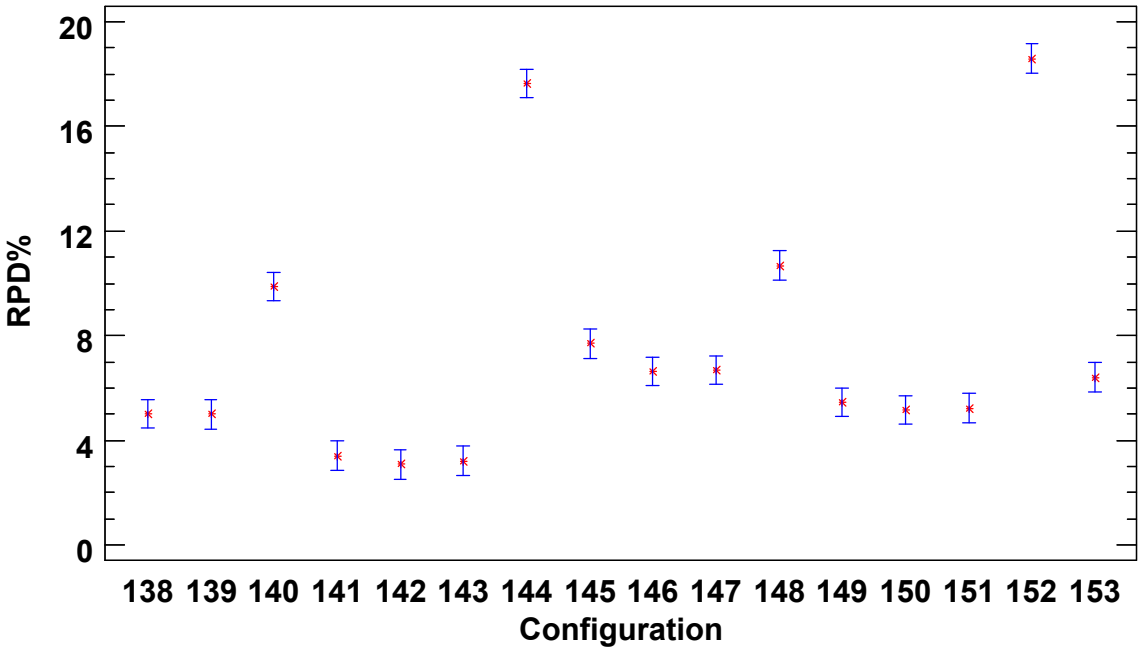


Figura 2.4. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 256 configurações de ILS_DP. Etapa 1 da calibração.

ANOVA, com nível de confiança de 95%. Como o *valor-p* é menor que 0,05 há diferença estatística entre as configurações testadas para o ILS_DP+PR.

Tabela 2.5. Tabela ANOVA referente a calibração de parâmetros do algoritmo ILS_DP+PR.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	15623,90	255	61,27	**429,95**	**0**
Resíduos	145,92	1024	0,14		
Total	15769,80	1279			

A Figura 2.5 mostra o gráfico de médias resultante do teste *Tukey* da Diferença Honestamente Significativa HSD (*Honestly Significant Difference*) com nível de confiança de 95% para as configurações testadas no algoritmo ILS_DP+PR. Somente as configurações de 120 a 132 são mostradas no gráfico. Nesta figura é possível ver que existe diferença significativa entre as configurações, pois há diversos intervalos que não se sobrepõe, mesmo embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. Para o algoritmo ILS_DP+PR a configuração 123 apresentou a melhor média e corresponde aos valores $N = 1$; $d_{max} = \lceil n/5 \rceil$; $sizeElite = 10$; $\gamma = 0{,}6$; $\beta = 0$ e $FFI = 1$. Como o valor encontrado para os parâmetros $d_{max} = \lceil n/5 \rceil$ e $sizeElite = 10$ são valores limites (dentre aqueles testados) visto que o maior valor de perturbação possível é $\lceil n/2 \rceil - 1$ e o menor valor

possível para $sizeElite = 1$. Assim, mais três níveis para d_{max} foram testados: $d_{max} \in \{\lceil n/4 \rceil; \lceil n/3 \rceil; \lceil n/2 \rceil - 1\}$. E mais 2 níveis para $sizeElite$: $sizeElite \in \{3; 5\}$. Para esta segunda etapa da calibração os melhores valores para os parâmetros foram: $d_{max} = \lceil n/3 \rceil$ e $sizeElite = 5$. Após as duas etapas de calibração os valores dos parâmetros utilizados nos experimentos finais para os algoritmos ILS_DP+PR foram $N = 1$; $d_{max} = \lceil n/3 \rceil$; $sizeElite = 5$; $\gamma = 0{,}6$; $\beta = 0$ e $FFI = 1$. Para mais detalhes dos testes estatísticos realizados o leitor pode consultar o Apêndice B.3.

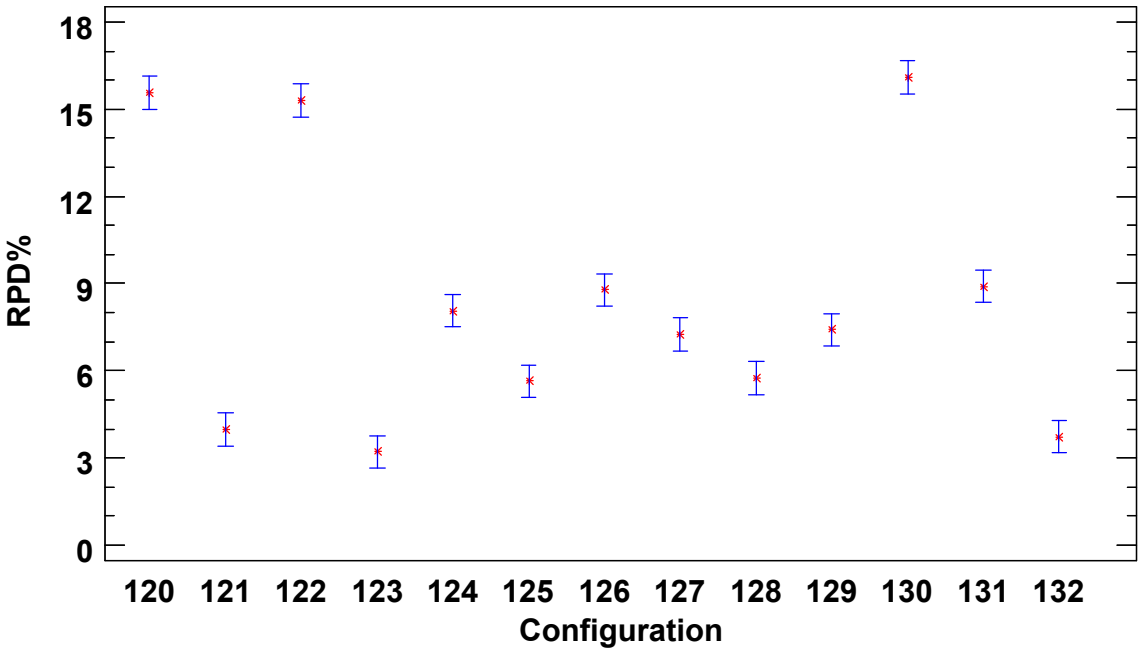


Figura 2.5. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 256 configurações de ILS_DP+PR. Etapa 1 da calibração.

A Tabela 2.6 mostra os valores finais após todas as etapas de calibração de cada algoritmo proposto.

Tabela 2.6. Valores finais dos parâmetros após a calibração.

Algoritmo	**Valores Escolhidos para os Parâmetros**
ILS_BASIC	$\gamma = 1{,}0$; $\beta = 0{,}3$; $FFI = 1$ e $d = \lceil n/3 \rceil$
ILS_DP	$\gamma = 0{,}6$; $\beta = 0{,}6$; $FFI = 1$; $N = 1$ e $d_{max} = \lceil n/3 \rceil$;
ILS_DP+PR	$\gamma = 0{,}6$; $\beta = 0{,}0$; $FFI = 1$; $N = 1$; $d_{max} = \lceil n/3 \rceil$ e $sizeElite = 5$

2.4.4 Resultados e comparações

Os resultados obtidos pelos algoritmos ILS_BASIC, ILS_DP, ILS_DP+PR e HGA são comparados usando o RDI (Equação (2.3)). Todos os algoritmos foram executados com o mesmo critério de parada que é baseado em uma quantidade de tempo de CPU, que foi fixada em $500 \times n$ milissegundos. Para resolver cada instância, cada algoritmo foi executado 30 vezes. O RDI é calculado considerando o valor da função objetivo em cada repetição.

A Tabela 2.7 mostra os resultados médios encontrados para todas as instâncias de problemas. As 1440 instâncias são agrupadas em 12 conjuntos de acordo com número de tarefas e número de famílias (veja coluna "Instância n x F" da Tabela 2.7). Esta tabela mostra o quanto a média das soluções de cada algoritmo difere da melhor solução conhecida. Claramente pode-se notar que os algoritmos propostos, ILS_BASIC, ILS_DP e ILS_DP+PR, apresentam melhores resultados em comparação ao HGA para todos os problemas testados. O algoritmo ILS_DP+PR em média apresenta um desempenho relativamente melhor que todos os outros, com RDI médio de 3,34.

Tabela 2.7. Índice de Desvio Relativo (RDI) Médio.

Instância n x F	**HGA**	**ILS_BASIC**	**ILS_DP**	**ILS_DP+PR**
60 x 2	12,70	1,93	1,98	**0,12**
60 x 3	26,58	1,91	2,33	**0,73**
60 x 4	35,48	3,04	3,26	**2,23**
60 x 5	38,62	4,94	5,73	**4,69**
80 x 2	22,48	4,40	4,15	**0,73**
80 x 3	33,87	5,39	4,38	**2,78**
80 x 4	43,48	5,38	5,32	**4,75**
80 x 5	46,86	6,74	**6,61**	7,00
100 x 2	29,03	5,35	3,79	**1,85**
100 x 3	41,42	7,03	5,55	**4,51**
100 x 4	45,14	4,87	**4,01**	4,02
100 x 5	47,43	7,01	7,00	**6,62**
Média	35,26	4,83	4,51	**3,34**

Para validar os resultados obtidos pelos quatro algoritmos e verificar se as diferenças observadas são estatisticamente significantes, foi realizada uma Análise de Variância (ANOVA) paramétrica. O *teste-F* na tabela ANOVA irá testar se há qualquer diferença significativa entre as médias do RDI (variável de resposta) encontradas pelos algoritmos (tratamentos). As hipóteses para o teste-F da ANOVA para os tratamentos são as seguintes:

- H_0: $m_{\text{HGA}} = m_{\text{ILS\_BASIC}} = m_{\text{ILS\_DP}} = m_{\text{ILS\_DP+PR}}$, ou, em outras palavras, todos os possíveis contrastes entre as médias dos algoritmos são estatisticamente nulos, no nível de significância considerado

- H_1: não H_0, ou, em outras palavras, existe pelo menos um contraste entre as médias dos algoritmos diferente de zero, no nível de significância considerado

Foram verificadas as três pressuposições da ANOVA para que os resultados do teste sejam estatisticamente válidos: normalidade (pelo teste de *Shapiro-Wilk W*), igualdade de variância (pelo teste de *Levene*) e a independência dos resíduos (pela plotagem dos resíduos). Os detalhes dos testes realizados encontram-se no Apêndice B.4.

O resultado da ANOVA pode ser visto na Tabela 2.8. O *teste-F* na tabela ANOVA irá testar se há qualquer diferença significante entre as médias do RDI (variável de resposta) encontradas por cada um dos quatro algoritmos (tratamentos). Esta tabela decompõe a variação total entre todas as observações na variação devido ao efeito dos tratamentos e na variação devida ao acaso ou resíduos (veja coluna "Fonte da Variação"). O número de graus de liberdade (veja coluna "Graus de Liberdade") dos tratamentos é igual a $I - 1$, onde I é o número de tratamentos utilizados. No caso, $I = 4$ algoritmos, sendo o número de graus de liberdade associado aos tratamentos igual a 3. Já o número de graus de liberdade associados aos resíduos é igual a $I(J - 1)$, onde J é o número de repetições. No caso, $J = 30$ repetições o que dá um total de 116 graus de liberdade. A coluna "Quadrados Médios" corresponde ao quociente entre a soma de quadrados pelo grau de liberdade da respectiva fonte de variação. O *valor de F*, que neste caso é igual a 35393,62, é o quociente entre o quadrado médio dos tratamentos pelo quadrado médio do resíduos. Uma vez que o *valor-P* do teste-F, que é o valor de interesse, é menor que 0,05, há uma diferença estatisticamente significante de um dos algoritmos para outro, com um nível de confiança de 95%.

Tabela 2.8. Tabela ANOVA referente a comparação do algoritmo HGA e dos algoritmos propostos ILS_BASIC, ILS_DP e ILS_DP+PR.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	21704,70	3	7234,91	**35393,62**	**0**
Resíduos	23,71	116	0,20		
Total	21728,40	119			

A ANOVA não especifica quais algoritmos são diferentes entre si, de modo que foi realizado um teste de Comparações Múltiplas para comparar cada par de médias

com um nível de confiança de 95%. A Tabela 2.9 mostra o resultado deste teste. A coluna "Diferença" mostra a média das amostras do primeiro algoritmo menos as do segundo. A coluna "+/- Limite" corresponde ao intervalo de incerteza para a diferença. Para qualquer par de algoritmos em que o valor absoluto da diferença exceda o limite significa que os algoritmos são significativamente diferentes no nível de confiança selecionado de 95%. Na tabela, a existência de diferença é indicado por um (*) na coluna "Significante". Pode-se ver que existe diferença estatisticamente significativa entre todos os pares de médias considerados. Estes resultados indicam, com um nível de confiança de 95%, que o algoritmo ILS_DP+PR é o melhor.

Tabela 2.9. Teste de comparações múltiplas para o RDI.

Par de algoritmos	**Significante**	**Diferença**	**+/-Limite**
HGA - ILS_BASIC	(*)	30,43	0,30
HGA - ILS_DP	(*)	30,75	0,30
HGA - ILS_DP+PR	(*)	31,92	0,30
ILS_BASIC - ILS_DP	(*)	0,32	0,30
ILS_BASIC - ILS_DP+PR	(*)	1,50	0,30
ILS_DP - ILS_DP+PR	(*)	1,17	0,30

A mesma análise pode ser vista na Figura 2.6. Esta figura mostra o gráfico de médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95%. Uma vez que o intervalo para o algoritmo HGA não sobrepõe os outros intervalos, a média do HGA é significativamente diferente das médias dos outros três algoritmos. Isto é, as heurísticas propostas mostram desempenho superior ao método HGA.

Para verificar mais claramente as diferenças entre as implementações propostas e com o intuito de analisar a eficiência da inclusão do controle automático da perturbação no algoritmo ILS_DP e a utilização do *path relinking* no ILS_DP+PR, foi realizada uma comparação somente entre ILS_BASIC, ILS_DP e ILS_DP+PR. Para esta comparação, o RDI é recalculado conforme (2.3) considerando somente estes três algoritmos. O teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95% mostrado na Figura 2.7 mostra com mais detalhes as diferenças entre as três implementações. Note como a variante ILS_DP+PR tem desempenho consideravelmente melhor que o ILS_DP, uma vez que não há sobreposição dos intervalos.

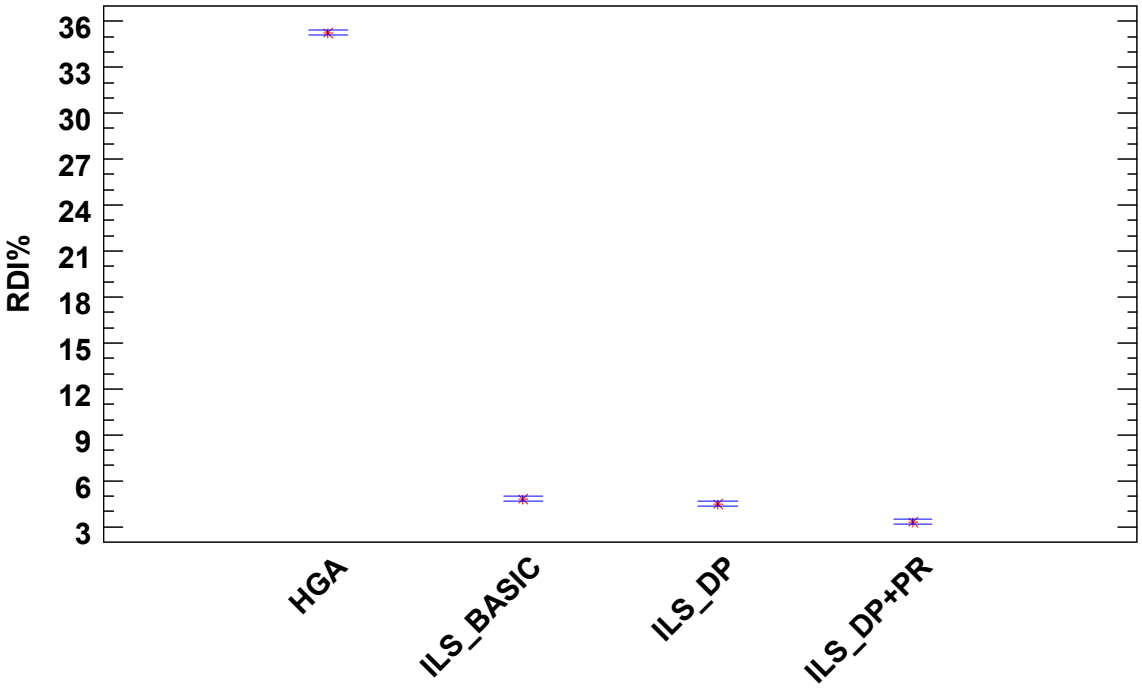


Figura 2.6. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do algoritmo HGA e dos algoritmos propostos ILS_BASIC, ILS_DP e ILS_DP+PR

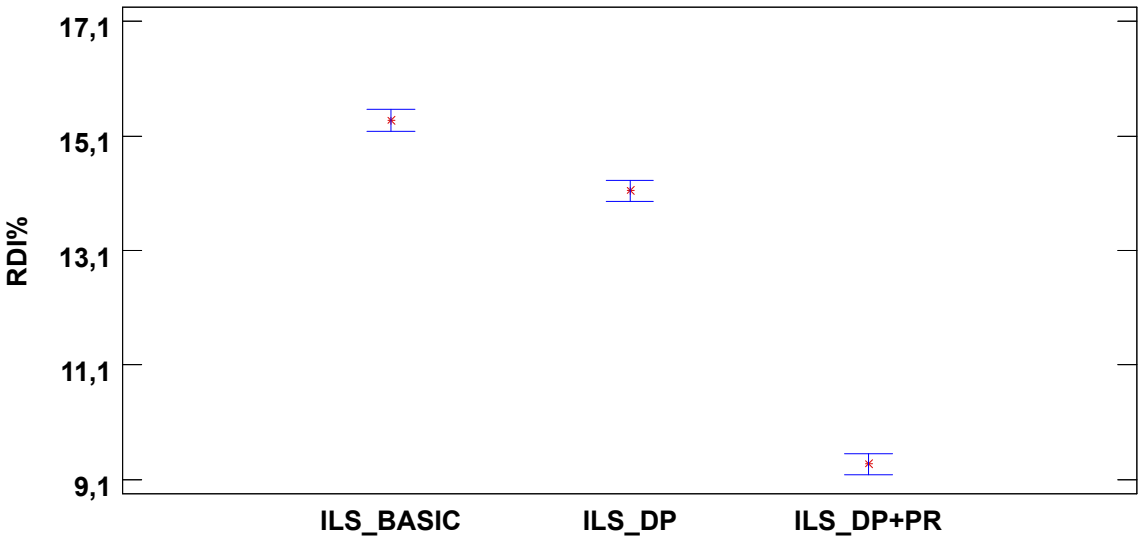


Figura 2.7. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos algoritmos ILS_BASIC, ILS_DP e ILS_DP+PR.

2.4.5 Resultados quanto aos parâmetros das instâncias

Na Seção 2.4.4 foram comparados os algoritmos quanto ao RDI médio considerando todas as classes de problemas. No entanto, é interessante também realizar uma

análise separada sobre cada parâmetro de modo que se possa verificar a influência deles sobre os resultados.

As Tabelas 2.10, 2.11, 2.12, 2.13 mostram, respectivamente, os resultados computacionais organizados por classes de problemas, considerando o número de tarefas, número de famílias, parâmetro de data de entrega e classes de *setup*. Por exemplo, a classe (60, *, *, *) representa todas as instâncias com 60 tarefas. Cada entrada é o RDI médio, considerando todas as instâncias de uma dada classe e o valor da função objetivo em todas as 30 execuções de um dado algoritmo.

O problema torna-se mais difícil quando o número de tarefas e famílias aumentam, embora as variantes ILS propostas deteriorarem mais devagar que HGA (veja Tabelas 2.10 e 2.11). É possível notar que o RDI médio de todos os algoritmos aumenta significando que a média das soluções encontradas por eles estão mais distante da melhor solução conhecida e, por conseguinte, mais distante da solução ótima. Observe na Tabela 2.10 que a medida que o número de tarefas aumenta de 60 para 100 todos os quatro algoritmos analisados tem valores maiores de RDI. Observe também que o RDI médio do algoritmo HGA varia de 28,34 para 40,76 enquanto que das variantes ILS propostas os valores são bem menores. O mesmo comportamento é observado só que considerando o número de famílias. Pela Tabela 2.11 é possível observar que a variante HGA degrada rapidamente à medida que o número de famílias aumenta de 2 para 5.

Tabela 2.10. Índice de Desvio Relativo (RDI) médio considerando somente classes de tarefas.

Classe de Problema	**HGA**	**ILS_BASIC**	**ILS_DP**	**ILS_DP+PR**
(60, *, *, *)	28,34	2,95	3,32	1,94
(80, *, *, *)	36,67	5,48	5,11	3,81
(100, *, *, *)	40,76	6,06	5,09	4,25
Média	35,26	4,83	4,51	3,34

Um resultado interessante é que o parâmetro associado a data de entrega influencia grandemente o algoritmo HGA. Os piores resultados do HGA ocorrem quando as datas de entregas são apertadas, principalmente pelas classes de problema (*, *, 0.5, *) quando a média do RDI foi 84,66 e a classe (*, *, 1.5, *) quando a média do RDI foi 45,61 (veja Tabela 2.12). Para estes casos, a variante ILS_DP+PR, por exemplo, se comporta bem com média do RDI de 4,54 e 4,45. Quando a data de entrega torna-se mais flexível, o que significa maiores valores do parâmetro associado a data de entrega, parece que o problema de torna mais fácil de resolver. Para as

Tabela 2.11. Índice de Desvio Relativo (RDI) médio considerando somente classes de famílias.

Classe de Problema	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
(*, 2, *, *)	21,40	3,90	3,31	0,90
(*, 3, *, *)	33,96	4,77	4,08	2,67
(*, 4, *, *)	41,37	4,43	4,20	3,67
(*, 5, *, *)	44,30	6,23	6,44	6,10
Média	35,26	4,83	4,51	3,34

classes (*, *, 2.5, *) e (*, *, 3.5, *) todos os algoritmos, inclusive o HGA, apresentam valores baixos de RDI. É interessante notar que a variante, ILS_BASIC apresenta os menores valores de RDI para as classes (*, *, 2.5, *) e (*, *, 3.5, *). Uma possível causa de a variante ILS_BASIC ter encontrado os melhores resultados de RDI para estas classes pode estar relacionado à configuração do parâmetro $\gamma = 1.0$, ou seja, a vizinha inteira é avaliada. Neste caso, a busca local se torna mais efetiva, já que a vizinhança é explorada exaustivamente, somando-se ao fato de para as classes (*, *, 2.5, *) e (*, *, 3.5, *) a data de entrega ser mais flexível.

Tabela 2.12. Índice de Desvio Relativo (RDI) médio considerando somente classes de data de entrega.

Classe de Problema	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
(*, *, 0.5, *)	84,66	10,90	8,27	4,54
(*, *, 1.5, *)	45,61	5,01	5,37	4,45
(*, *, 2.5, *)	8,98	2,75	3,05	3,24
(*, *, 3.5, *)	1,78	0,66	1,34	1,11
Média	35,26	4,83	4,51	3,34

O outro parâmetro é relacionado a classes de *setup*. Note que este parâmetro tem pouca influência na performance das implementações ILS propostas (veja Tabela 2.13), onde há pouca variação na média do RDI. A variante HGA no entanto degrada rapidamente entre as classes de *setup*, com pior desempenho para a classe de setup L.

2.4.6 Resultados para instâncias de pequeno porte

Um conjunto com 216 instâncias de pequeno porte foram geradas. Estas instâncias consideram $n \in \{10, 15, 20\}$; $F \in \{2, 4, 6\}$; $r \in \{0.5, 1.5, 2.5, 3.5\}$ e classes de *setup*

Tabela 2.13. Índice de Desvio Relativo (RDI) médio considerando somente classes de *setup*.

Classe de Problema	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
(*, *, *, S)	22,46	5,34	4,21	2,43
(*, *, *, M)	36,55	3,59	3,65	2,69
(*, *, *, L)	46,76	5,57	5,67	4,88
Média	35,26	4,83	4,51	3,34

$\in \{S, M, L\}$. Duas instâncias foram geradas para cada uma das 108 configurações. Os tempos de processamento e as datas de entrega foram gerados da mesma forma como mostrado na Seção 2.4.1.

Para as instâncias de pequeno porte, a comparação é feita entre as soluções encontradas pelo algoritmo ILS_DP+PR, que mostrou ser a melhor implementação, e a solução encontrada pela execução de um modelo matemático de Programação Inteira Mista (MIP) no *solver* ILOG/CPLEX versão 12.4. Cada instância foi rodada uma única vez no *solver* durante um tempo de 30 minutos. O algoritmo ILS_DP+PR é executado 30 vezes para cada problema teste.

O modelo matemático é o mesmo apresentado no trabalho de Chantaravarapan et al. (2003)[1]. A máquina utilizada nos experimentos possui um processador Intel(R) Xeon(R) CPU X5650 que possui 12 núcleos, sendo utilizados todos eles na resolução do modelo.

Primeiramente, é feita uma análise considerando-se a melhor solução encontrada nas 30 execuções do algoritmo ILS_DP+PR. Os resultados para este experimento são sumarizados na Tabela 2.14. Para instâncias com 10 e 15 tarefas, o CPLEX conseguiu provar todos os 144 ótimos, que também foram encontrados pelo algoritmo ILS_DP+PR. Para as instâncias com 20 tarefas, o CPLEX conseguiu provar o ótimo em 57 delas. Para as 15 instâncias com 20 tarefas, que não tiveram o ótimo provado, em 10 delas o ILS_DP+PR e o CPLEX encontraram soluções de mesma qualidade. Nas outras 5 instâncias, o algoritmo ILS_DP+PR encontrou soluções de melhor qualidade que aquelas fornecidas pelo CPLEX. A Tabela 2.15 mostra os resultados para estas instâncias que o ILS_DP+PR encontrou soluções melhores que o CPLEX. Para estas instâncias o algoritmo ILS_DP+PR encontrou soluções 11,40% melhores que o CPLEX.

Das 216 instâncias testadas em nenhum caso a melhor solução do algorimo ILS_DP+PR foi pior que a solução obtida pelo CPLEX. Dos 201 ótimos provados

[1]O modelo matemático utilizado pode ser consultado no apêndice A.

Tabela 2.14. Comparação do número de soluções ótimas encontradas pelo CPLEX e ILS_DP+PR, considerando somente instâncias de pequeno porte.

n	Número de instâncias	Ótimos Provados pelo CPLEX	Nº Soluções do ILS_DP+PR Iguais as do CPLEX	Nº Soluções do ILS_DP+PR Melhores que do CPLEX
10	72	72	72	0
15	72	72	72	0
20	72	57	67	5
Total	216	201	211	5

Tabela 2.15. Instâncias de pequeno porte em que ILS_DP+PR encontrou soluções melhores que o CPLEX durante um tempo de execução de 30 minutos. f_{ILS_DP+PR} é o melhor valor da função objetivo encontrado pelo algoritmo ILS_DP+PR e f_{CPLEX} é o limite superior fornecido pelo CPLEX no tempo estabelecido.

Instância	$\mathbf{f_{ILS\_DP+PR}}$	$\mathbf{f_{CPLEX}}$	$\mathbf{GAP_{CPLEX}}$	$\frac{\mathbf{f_{ILS\_DP+PR}} - \mathbf{f_{CPLEX}}}{\mathbf{f_{CPLEX}}} \times \mathbf{100}\%$
20-4-1.5-M-0	710	826	95,07%	-14,04%
20-6-0.5-M-1	7458	7472	27,72%	-0,19%
20-6-1.5-M-1	965	1125	100,00%	-14,22%
20-6-2.5-L-0	2589	3272	89,63%	-20,87%
20-6-2.5-L-1	1965	2128	72,43%	-7,66%
Média				-11,40%

pelo CPLEX, ILS_DP+PR encontrou as mesmas soluções.

Outra análise foi feita, mas considerando agora a solução média (não mais a melhor solução) encontrada pelo algoritmo ILS_DP+PR. Para esta análise são consideradas todas as 216 instâncias de pequeno porte. O RDI% é calculado conforme equação (2.3), sendo f_{best} e f_{worst}, respectivamente, a melhor e a pior solução encontrada; f_{method} é a solução média encontrada pelo algoritmo ILS_DP+PR. Os resultados são mostrados na Tabela 2.16. Como pode ser visto nesta tabela, o RDI médio para o algoritmo ILS_DP+PR foi de 0,10%, o que mostra que, na média, as soluções encontradas pelo ILS_DP+PR são muito próximas da melhor solução conhecida. Somente para a classe de problemas com 10 tarefas e 4 famílias (10 x 4) a média das soluções do ILS_DP+PR diferem da melhor solução conhecida. Nas demais classes o RDI médio foi de 0,00%, o que indica que, para estas classes, o ILS_DP+PR encontrou a melhor solução na maioria das repetições.

Tabela 2.16. Índice de Desvio Relativo (RDI) considerando a média das soluções encontradas pelo ILS_DP+PR.

Instância $n \times F$	**RDI%**
10 x 2	0,00
10 x 4	0,86
10 x 6	0,00
15 x 2	0,00
15 x 4	0,00
15 x 6	0,00
20 x 2	0,00
20 x 4	0,00
20 x 6	0,00
Média	0,10

2.4.7 Análise de tempo

Os experimentos descritos anteriormente foram feitos com o foco na qualidade das soluções com respeito ao valor da função objetivo. No entanto, é importante considerar também o desempenho dos algoritmos quanto ao tempo de execução. Para este fim, foram realizados testes de probabilidade empírica juntamente com *time-to-target plots*. Estas duas ferramentas tem sido utilizadas no desenvolvimento de algoritmos e como estratégia de comparações de implementações (Haddad, 2012), sejam elas diferentes heurísticas, implementações paralelas usando um número diferente de processadores ou um mesmo algoritmo usando diferentes estratégias ou configurações de parâmetros.

Time-to-target plots mostram no eixo das ordenadas a probabilidade de que um determinado algoritmo encontre uma solução tão boa quanto uma determinada solução alvo dentro de um tempo de execução (mostrado no eixo das abcissas). *Time-to-target plots* foram utilizados como estratégia de análise e comparação em diversos trabalhos de otimização combinatória como nos trabalhos de Feo et al. (1994) e Aiex et al. (2002).

Para se realizar este experimento, foi escolhida uma instância com 100 tarefas e 4 famílias, onde o valor alvo foi encontrado por todas as 4 implementações pelo menos 1 vez. O valor alvo é o melhor valor da função objetivo encontrado para esta instância. Cada algoritmo é aplicado $n = 100$ vezes e sempre que o valor alvo é alcançado o tempo de execução (em segundos) é registrado e o algoritmo é interrom-

pido. Os tempos de execução são, então, ordenados de forma não decrescente. Para cada tempo de execução t_i, registrado por um determinado algoritmo, é calculada a probabilidade cumulativa p_i, de acordo com a equação (2.4). O gráfico (*time-to-target plots*) gerado é apresentado na Figura 2.8 e corresponde a comparação entre os algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR. Observe que as curvas foram sobrepostas para fique mais fácil a visualização das curvas de probabilidades empíricas.

$$p_i = \frac{(i - 0{,}5)}{n}, \forall i = 1, 2, \cdots, n \tag{2.4}$$

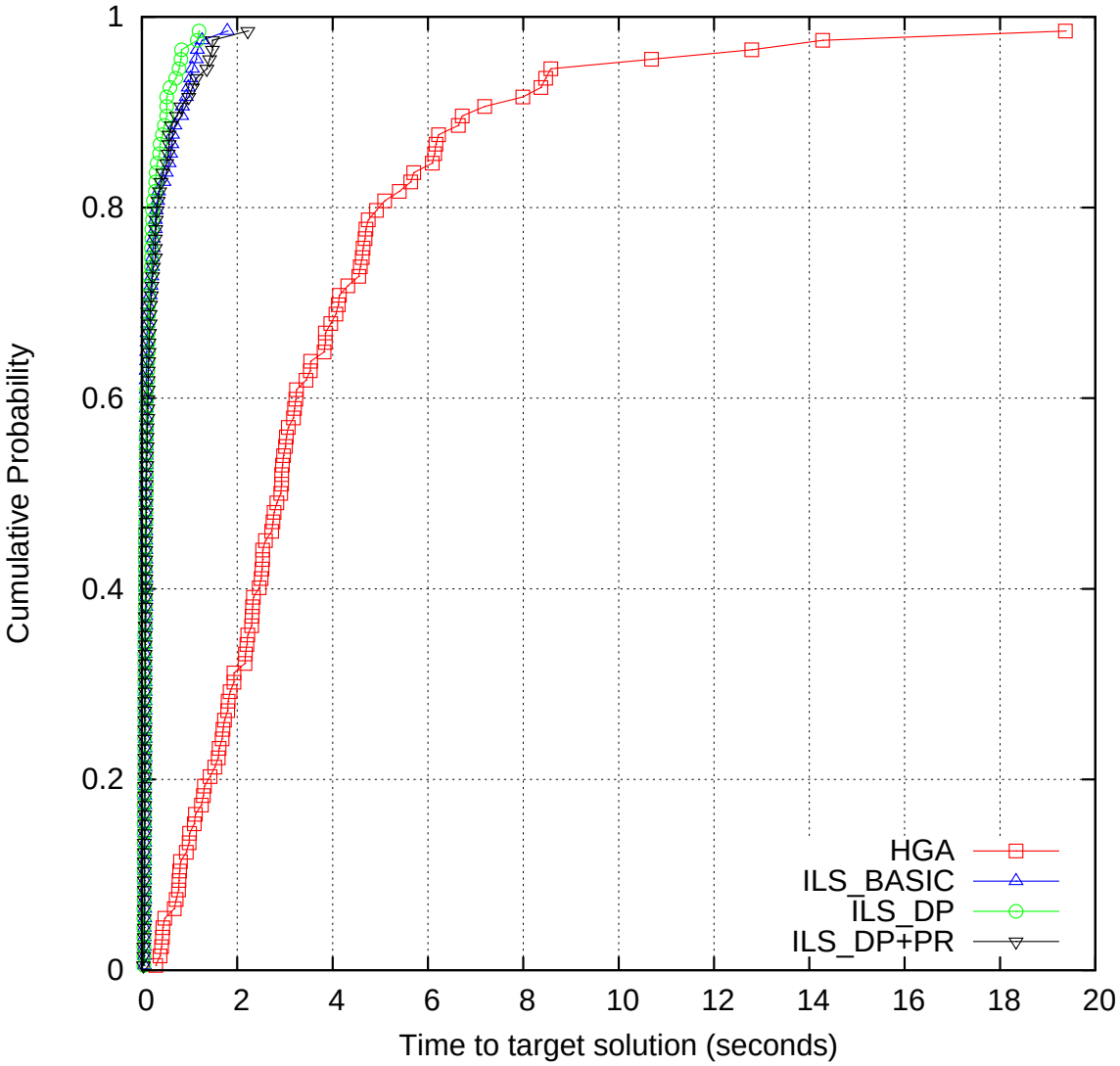


Figura 2.8. Gráfico de distribuição de probabilidades cumulativas dos algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR - instância com 100 tarefas e 4 famílias.

Ao analisar o gráfico, é fácil observar que as 3 implementações propostas de ILS encontram a solução alvo mais rapidamente que o HGA. Nota-se também que o algoritmo ILS_DP tem um comportamento um pouco melhor que as implementações ILS_BASIC e ILS_DP+PR, tendo uma maior probabilidade de encontrar a solução alvo mais rapidamente. Nota-se também que a implementação ILS_DP+PR possui comportamento similar ao algoritmo ILS_BASIC.

No entanto, é necessário validar estes resultados através de métodos estatísticos. Para este fim, foi utilizada a ferramenta numérica, proposta nos trabalhos realizados por Ribeiro & Rosseti (2009) e Ribeiro et al. (2012), que compara qualquer par de algoritmos de busca local estocásticos. Esta ferramenta calcula a probabilidade de que um determinado algoritmo A_1 encontre uma solução tão boa quanto uma solução alvo em um tempo menor que outro algoritmo A_2. Denota-se por X_1 e X_2 a variável aleatória contínua representando o tempo necessário para os algoritmos A_1 e A_2, respectivamente, encontrarem uma solução tão boa quanto um alvo para um determinado problema teste. O objetivo é determinar a probabilidade, $Pr(X_1 \leq X_2)$, que X_1 tenha um valor menor ou igual a $X2$, o que, em outras palavras, significa que o algoritmo A_1 possui performance melhor que A_2.

Primeiramente, é feita a análise com relação às três implementações ILS propostas. A Figura 2.9 mostra a distribuição de probabilidades entre os algoritmos ILS_BASIC e ILS_DP. Sendo A_1 o algoritmo ILS_DP e A_2 o algoritmo ILS_BASIC, a probabilidade calculada de o algoritmo ILS_DP encontrar uma solução tão boa quanto o alvo em um tempo inferior ao algoritmo ILS_BASIC foi de $Pr(A_1 \leq A_2) = 55{,}92\%$ com um erro de 6,50%.

A Figura 2.10 mostra os resultados do teste de probabilidade empírica para os algoritmos ILS_DP+PR (A_1) e ILS_BASIC (A_2). A probabilidade calculada considerando este par foi: $Pr(A_1 \leq A_2) = 50{,}28\%$ com um erro de 6,82%, o que mostra que estes dois pares de algoritmos tiveram comportamento similar. Considerando o par de algoritmos ILS_DP+PR (A_1) e ILS_DP (A_2), a probabilidade calculada foi $Pr(A_1 \leq A_2) = 44{,}70\%$ com um erro de 6,21%, o que indica que a versão ILS_DP é melhor, como também pode ser visto na Figura 2.11.

Por fim, é feita a comparação entre a versão ILS_DP, que mostrou ser a melhor no teste de probabilidade empírica dentre as implementações ILS propostas, e a implementação HGA. O gráfico da distribuição de probabilidades deste par de algoritmos é mostrado na Figura 2.12. Para este par, a probabilidade calculada foi $Pr(A_1 \leq A_2) = 98{,}88\%$ com um erro de 0,06%, onde A_1 é ILS_DP e A_2 é HGA, mostrando a superioridade do algoritmo ILS_DP.

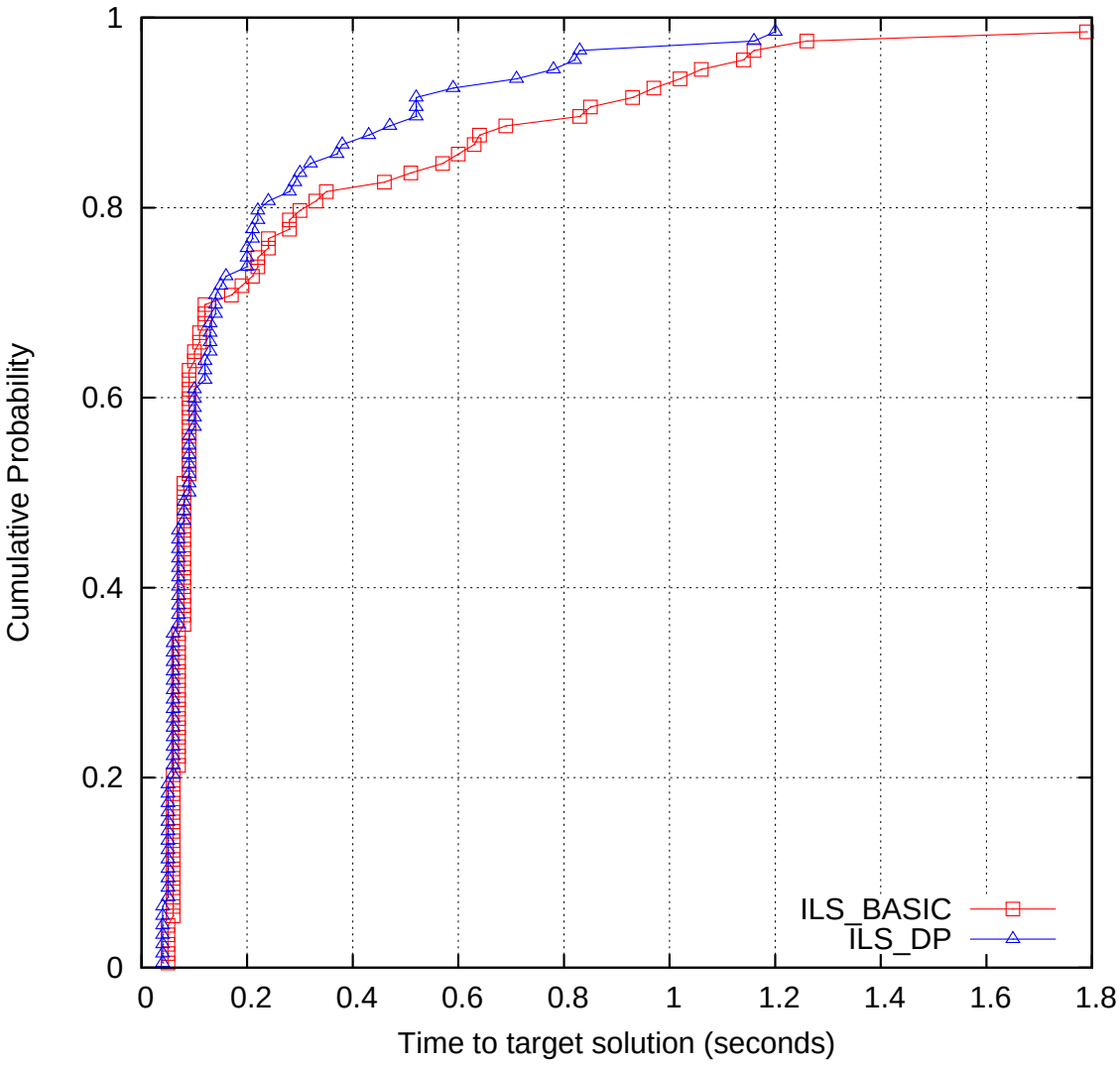


Figura 2.9. Gráfico de distribuição de probabilidades cumulativas entre os algoritmos ILS_BASIC e ILS_DP - instância com 100 tarefas e 4 famílias.

2.5 Conclusões

A proposta deste estudo foi examinar o problema de programação de tarefas em uma máquina com tempo de preparação dependente da sequência das famílias com o objetivo de minimizar o atraso total. O problema $1|ST_{sd,b}|\sum T_j$ é um problema complexo de otimização combinatória com ampla aplicação nas indústrias. Em virtude da natureza $\mathcal{NP}$-Difícil deste tipo de problema, três heurísticas baseadas na metaheurística *Iterated Local Search* (ILS), que mantém tanto simplicidade quando generalidade, são propostas. Também foi testada uma estratégia de controle automático da força de perturbação e a técnica de intensificação *Path Relinking*. Os parâmetros dos algoritmos foram analisados e determinados através da metodologia

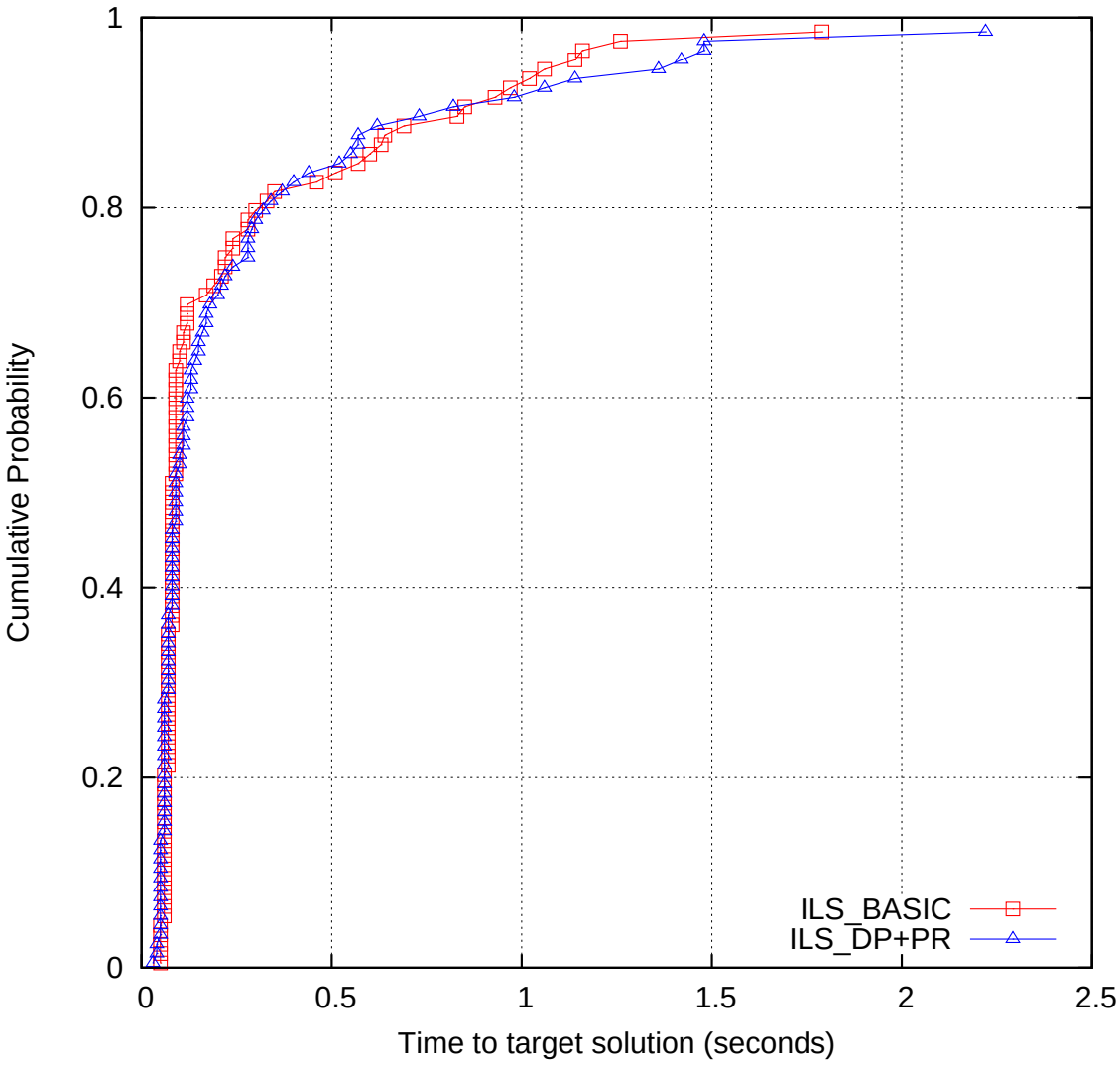


Figura 2.10. Gráfico de distribuição de probabilidades cumulativas entre os algoritmos ILS_BASIC e ILS_DP+PR - instância com 100 tarefas e 4 famílias.

Desenho de Experimentos e pela Análise de Variância.

Para avaliar a aplicabilidade das heurísticas ILS_BASIC, ILS_DP e ILS_DP+PR propostas, seus desempenhos foram comparados com o melhor algoritmo (algoritmo genético hibrido HGA) disponível na literatura em um grande conjunto de problemas testes. Os resultados computacionais indicam que a inclusão do controle automático da perturbação e a utilização de *Path Relinking* no ILS produz um melhoramento significativo no algoritmo, mostrando serem viáveis a utilização das duas técnicas. Os resultados obtidos também mostram que as heurísticas propostas são mais efetivas que o algoritmo HGA.

A análise quanto ao tempo de execução e convergência dos algoritmos foram

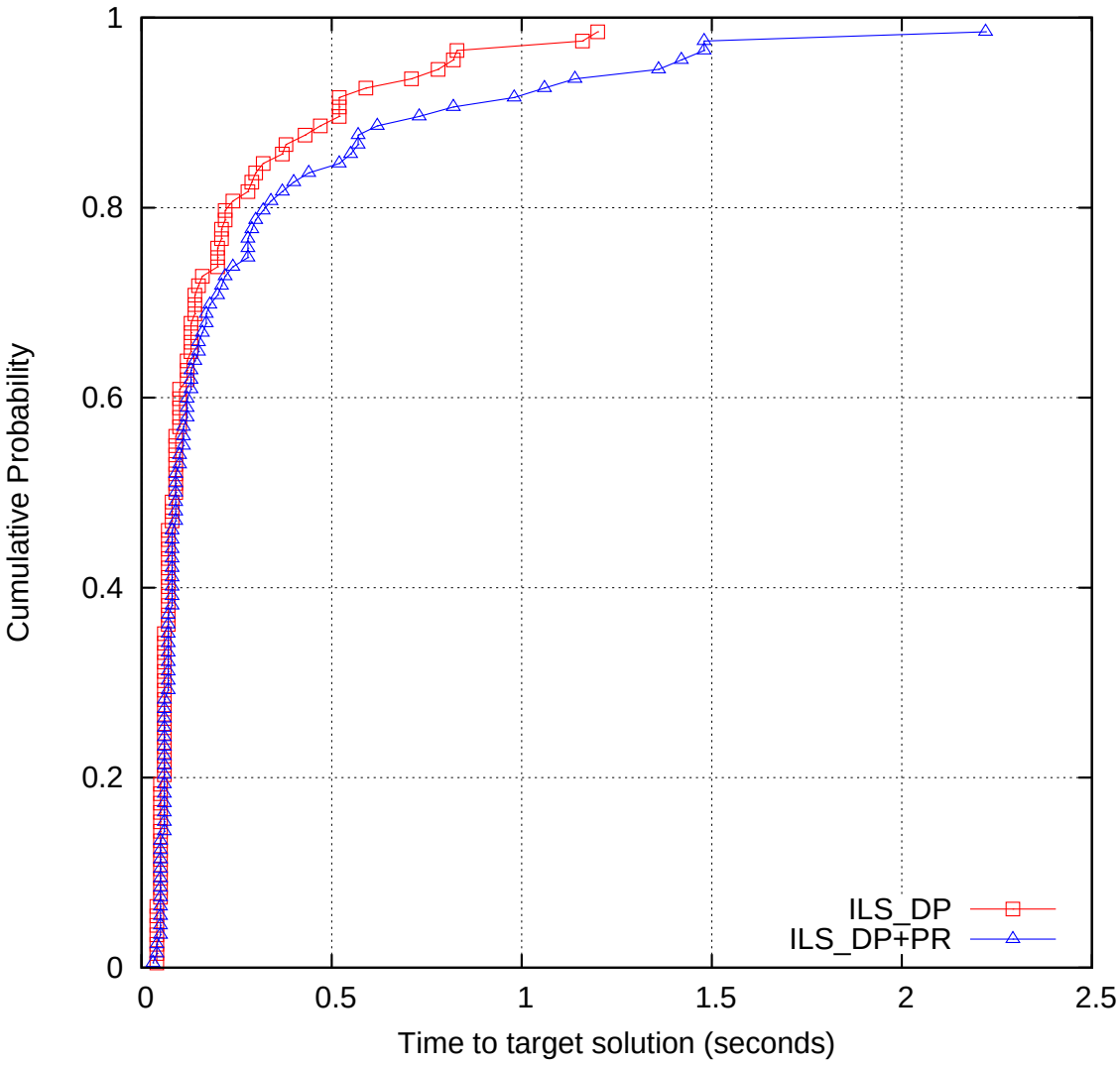


Figura 2.11. Gráfico de distribuição de probabilidades cumulativas entre os algoritmos ILS_DP e ILS_DP+PR - instância com 100 tarefas e 4 famílias.

realizadas através de testes de probabilidades empíricas, indicando superioridade da versão ILS_DP sobre as outras.

A comparação com um modelo matemático de programação linear inteira mista também foi realizada utilizando um conjunto de 216 instâncias de pequeno porte. Os resultados indicam que a variante ILS_DP+PR proposta apresenta um *gap* muito pequeno quanto às soluções ótimas, sendo que a heurística proposta encontrou o valor ótimo em 201 das 216 instâncias.

Os resultados computacionais foram validados através de análises estatísticas, mostrando que a variante ILS_DP+PR é o melhor algoritmo quanto a qualidade das soluções e a variante ILS_DP é o melhor algoritmo quanto ao tempo de execução.

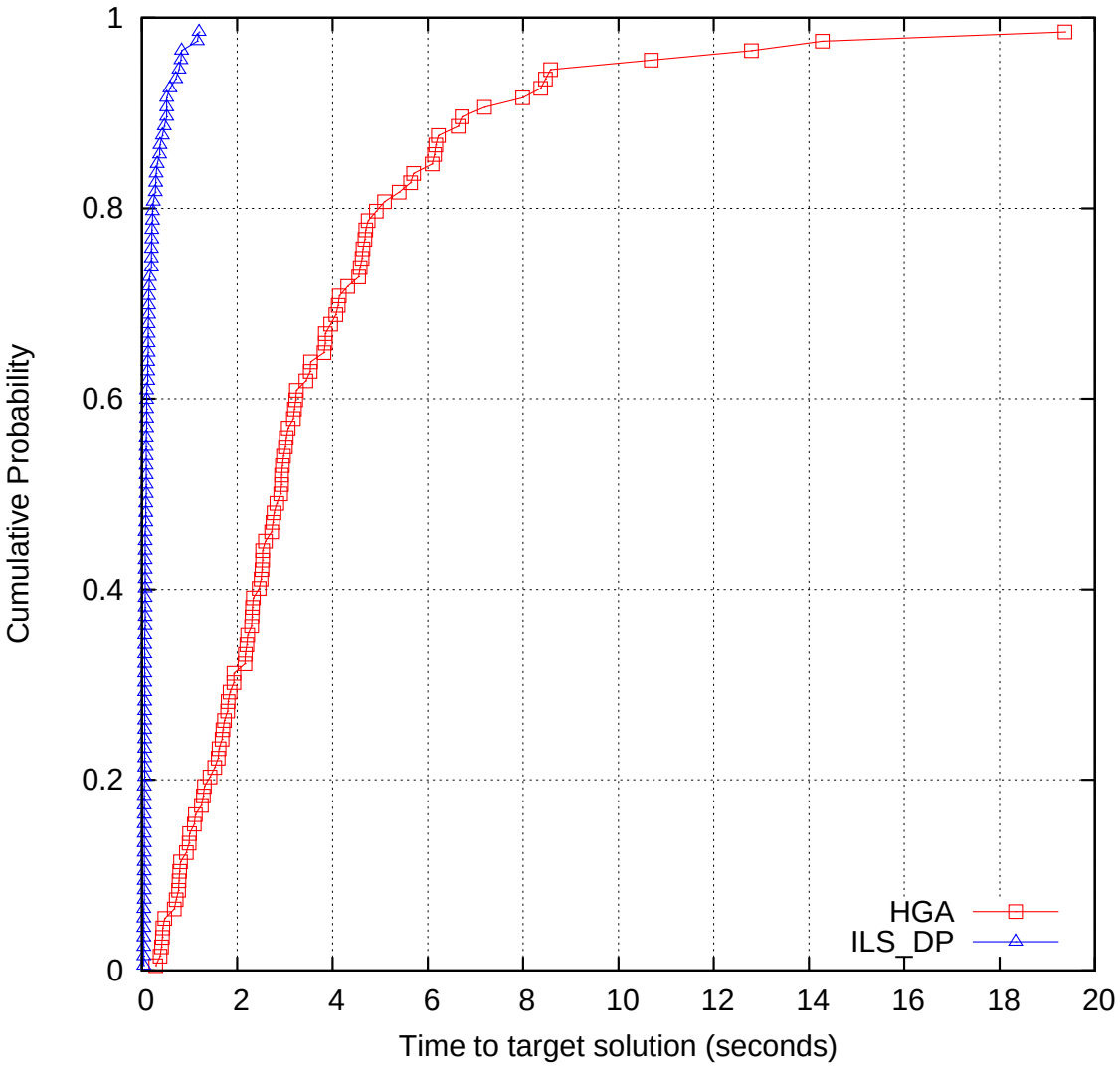


Figura 2.12. Gráfico de distribuição de probabilidades cumulativas entre os algoritmos ILS_DP e HGA - instância com 100 tarefas e 4 famílias.

Algumas questões ainda parecem ser valorosos objetos de estudos. Por exemplo, considerar um ambiente mais complexo de máquinas paralelas independentes onde o tempo de preparação dependa, além das famílias, de qual máquina a tarefa será executada. Para estes casos, as heurísticas desenvolvidas neste trabalho poderiam ser utilizadas, sendo necessário testar outras buscas locais mais eficientes quanto ao tempo de execução. Na atualização do conjunto elite, no algoritmo ILS_DP+PR, também poderia ser utilizado um critério de inserção baseado num valor limiar, calculado de acordo com o número de diferenças com respeito às soluções que já estão no conjunto. Considerar também o atraso total ponderado parece ser adequado. Avaliar a utilização da metaheurística *Iterated Greedy* (IG) que tem

obtido bons resultados para problemas de sequenciamento também pode melhorar os resultados.

Parte II

O problema $1|r_{ij}|\Sigma w_{ij}F_{ij} + \Sigma\delta_i d_i$

Capítulo 3

Problema de sequenciamento em uma máquina com custos de entrega de lotes

Neste capítulo, será tratado especificamente de um problema de sequenciamento da produção em uma máquina no qual as tarefas finais (pedidos) são entregues aos clientes em lotes, sendo que a formação dos lotes está associada aos custos de entrega. O problema consiste em determinar a sequência de processamentos das tarefas e os lotes de tarefas de cada cliente de tal maneira que o fluxo ponderado total mais os custos totais de entrega sejam minimizados. Primeiramente, na Seção 3.1, é feita uma introdução sobre o problema, seguido de uma revisão de literatura na Seção 3.2. Um modelo matemático de Programação Linear Inteira Mista é proposto e apresentado na Seção 3.3. As heurísticas propostas são explicadas na Seção 3.4. Os experimentos computacionais e os resultados alcançados estão organizado na Seção 3.5. Finalmente, na Seção 3.6, estão as conclusões e trabalhos futuros para este segundo problema abordado.

3.1 Introdução

Uma cadeia de fornecimento de um produto típico começa pela chegada do material, seguido da produção propriamente dita e por fim a distribuição do produto final aos clientes. Segundo Hall & Potts (2003), uma cadeia de fornecimento representa todos os estágios nos quais algum valor é adicionado a um produto manufaturado, incluindo o fornecimento de matéria-prima e componentes intermediários, fabricação

de produtos acabados, empacotamento, transporte, armazenagem e logística.

Neste contexto, existem quatro tipos de decisões principais a serem tomadas. Primeiro, as **decisões de localização**, que basicamente procuram determinar a localização geográfica das facilidades, pontos de estoque, acesso ao mercado consumidor, custos de produção, impostos, custos de distribuição, dentre outros. Depois, **decisões de produção**, que devem levar em consideração quais produtos produzir e em quais facilidades eles devem ser produzidos, o fluxo exato de produção entre as facilidades, capacidade de produção das facilidades e, principalmente, realizar o cronograma de produção mestre ou sequenciamento da produção. **Decisões de inventário** se referem a como os estoques serão gerenciados, uma vez que estes existem em basicamente todos as etapas de uma cadeia de fornecimento. Por fim, devem ser consideradas as **decisões de transporte**, que estão fortemente relacionadas a decisões de inventário. O nível de serviço que se deseja oferecer aos clientes e sua localização geográfica bem como a forma como serão entregues os produtos também são fatores que fortemente influenciam neste tipo de decisão (Ganeshan & Harrison, 1995).

Tradicionalmente, as operações de *marketing*, distribuição, planejamento, manufatura e organização de compras ao longo da cadeia de fornecimento de um produto eram realizadas de forma independente (Ganeshan & Harrison, 1995). Porém, o acirramento da concorrência entre as empresas num âmbito global as tem forçado a investirem maciçamente na redução de custos (de estoques, por exemplo) e melhoria dos serviços oferecidos aos seus clientes, tal que os prazos devem ser rigorosamente cumpridos. Estas necessidades impostas pela ampla concorrência levam, então, as empresas a considerarem não somente a otimização de etapas individuais na cadeia de fornecimento, mas também a coordenação entre diversas operações de forma a obterem maior eficiência no processo produtivo.

Um tipo de coordenação que pode ocorrer, e que tem recebido atenção em logística e gerenciamento da manufatura, é entre o sequenciamento da produção e transporte, principalmente com a tendência de diminuição de estoques. Um produto manufaturado (ou semimanufaturado), algumas vezes, deve ser transportado de uma área de espera para alguma outra instalação de manufatura para futuro processamento ou, então, ser entregue aos clientes. Este tipo de coordenação pode fazer com que as empresas de manufatura economizem dinheiro e combustível, o que é muito importante tanto do ponto de vista econômico quanto do ponto de vista ambiental.

Segundo Potts & Kovalyov (2000), os produtos (semi) manufaturados, ou simplesmente tarefas, como será mencionado daqui em diante, podem ser caracterizados

quanto à disponibilidade após o seu processamento. No primeiro caso, dito ser de disponibilidade de tarefas (*job availability*), as tarefas estão disponíveis individualmente, para serem entregues aos clientes ou para a próxima estação de processamento, logo após terem sido processadas. O outro caso, dito ser de disponibilidade de lote (*batch availability*), as tarefas em um mesmo lote ficam disponíveis somente após todas as outras do mesmo lote terem sido processadas. Ou, em outras palavras, todas as tarefas de um mesmo lote são completadas juntas quando a última tarefa do lote termina o seu processamento.

Nesta segunda parte do trabalho é tratado um problema de sequenciamento da produção, em uma máquina, que envolve tanto o sequenciamento de tarefas quanto a formação de lotes para o atendimento dos clientes. É considerada também a disponibilidade de lotes. Formalmente, o problema é descrito a seguir.

Há um conjunto de N tarefas a ser sequenciado em uma máquina, sem interrupção ou preempção. A máquina pode processar no máximo uma única tarefa por vez. As N tarefas pertencem a pedidos de F clientes, de modo que cada cliente i ($\forall i = 1, 2, \cdots, F$) possui n_i tarefas. Denota-se por J_{ij} a j-ésima tarefa do cliente i. Cada tarefa J_{ij} possui um tempo de processamento (p_{ij}), uma data de liberação (r_{ij}), que representa o instante a partir do qual a tarefa está apta a ser processada, e um peso ou prioridade (w_{ij}).

Neste problema, assim que uma tarefa termina o seu processamento, ou ela pode ser entregue imediatamente ou então esperar para ser entregue em um lote junto com outra(s) tarefa(s) do mesmo cliente. Neste último caso, o tempo de fluxo da tarefa J_{ij}, denotado por F_{ij}, é definido como o instante a partir do qual o lote está pronto para ser entregue e será determinado pelo tempo de conclusão da última tarefa pertencente ao mesmo lote da tarefa J_{ij}. Há um custo não-negativo de entrega, d_i, para cada lote entregue ao cliente i ($\forall i = 1, 2, \cdots, F$). O objetivo do problema é encontrar um sequenciamento de tarefas e determinação dos lotes que serão entregues aos clientes tal que o fluxo total ponderado mais os custos de entrega seja mínimo. O problema é denotado por $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, seguindo a notação de três campos proposta por Graham et al. (1979), onde δ_i é o número de lotes entregues ao cliente i e é um valor a ser determinado.

O fluxo ponderado pode ser interpretado como custos de espera (ou custos de estoques). Os custos de entrega estão associados a transferir as tarefas de um sistema para outro, ou então, entregá-las aos clientes. A entrega realizada através de lotes de tarefas pode ajudar a economizar os custos de entrega (Selvarajah et al., 2013). Existe um *trade-off* a ser tratado: entregar cada tarefa individualmente pode aumentar os custos associados a entregas (acarreta um aumento na parcela $\sum \delta_i d_i$

da função objetivo) ao passo que colocar várias tarefas em um mesmo lote pode aumentar o tempo de fluxo ponderado (acarreta um aumento na parcela $\sum w_{ij}F_{ij}$ da função objetivo). Os pesos e a data de liberação das tarefas são complicadores a mais já que irão influenciar tanto no sequenciamento das tarefas quando na formação dos lotes.

Uma vez que o problema de minimização do fluxo ponderado total com custos de entrega sem data de liberação, $1||\sum w_{ij}F_{ij}+\sum \delta_i d_i$, pertence à classe de problema $\mathcal{NP}$-Difícil mesmo quando o número de lotes é constante (Ji et al., 2007), segue que o problema $1|r_{ij}|\sum w_{ij}F_{ij}+\sum \delta_i d_i$ também pertence a $\mathcal{NP}$-Difícil (Mazdeha et al., 2012).

O restante das seções deste capítulo está organizado como segue. Na Seção 3.2 é feita uma revisão da literatura dos trabalhos mais relacionados ao problema abordado. Na Seção 3.3 é proposta uma formulação matemática de Programação Linear Inteira Mista (MILP) para resolver o problema de forma exata. Na Seção 3.4 são descritos com detalhes as heurísticas propostas e na Seção 3.5 estão os experimentos computacionais realizados. Finalmente, as conclusões são dadas na Seção 3.6.

3.2 Revisão de literatura

O problema de sequenciamento para a minimização do tempo de fluxo ponderado total com considerações de tempo de liberação de tarefas tem sido bastante estudado na literatura.

A forma mais básica do problema, isto é, quando não se leva em consideração a data de liberação das tarefas, foi trabalhada por Smith (1956) que provou que a regra de despacho SPT (*Shortest Processing Time*) pode ser utilizada para se resolver de forma ótima o problema de minimização do tempo de fluxo total, denominado, de acordo com a notação de três campos de Graham et al. (1979), por $1||\sum F_j$. Smith (1956) também provou que a regra WSPT (*Weighted Shortest Processing Time*) pode ser utilizada para se resolver a versão ponderada do problema ($1||\sum F_j w_j$), onde w_j é o peso ou prioridade de uma determinada tarefa j.

A consideração da data de liberação (r_j), no entanto, é importante do ponto de vista prático, porque em um sistema de produção real representa o instante em que uma tarefa chega, ou torna-se disponível, no sistema produtivo. Lenstra et al. (1977) demostrou que a versão com uma data de liberação diferente para cada tarefa, mesmo quando não se considera pesos, $1|r_j|\sum F_j$, é um problema fortemente $\mathcal{NP}$-

Difícil. O problema $1|r_j|\sum F_j$ também foi abordado por Chandra (1979), Dessouky & Deogun (1981) e Chu (1992) que propuseram heurísticas e algoritmos *Branch and Bound* (B&B), sendo que este último trabalho conseguiu resolver problemas com até 100 tarefas. A versão ponderada deste mesmo problema, $1|r_j|\sum w_j F_j$, foi trabalhada por Bianco & Ricciardelli (1982), Hariri & Potts (1983) e Belouadah et al. (1992) utilizando-se também de algoritmos B&B, sendo que os dois últimos conseguiram resolver instâncias com até 50 tarefas.

Em situações reais de produção, existem custos ao se transferir as tarefas que acabaram de serem processadas (ou semimanufaturadas) para outra etapa do processo produtivo ou então serem entregues aos devidos clientes. Estes custos de entrega normalmente estão associadas a transferir uma tarefa de um local para outro. Por questão de economia na entrega, pode ser interessante que estas tarefas entregues a um mesmo destino sejam transferidas em lotes, que contenham mais de uma tarefa. Havendo a possibilidade de formação de lote, uma tarefa que termina o seu processamento pode ou ser entregue imediatamente ou ficar aguardando para ser entregue junto com outras no mesmo lote, aumentando, assim, o seu tempo de conclusão que será definido como tempo de conclusão da última tarefa processada do lote. No primeiro caso, o custo de entrega aumenta, enquanto no outro o tempo de fluxo aumenta. Portanto, problemas que consideram tanto o sequenciamento das tarefas quando a formação de lotes e custos de entrega tornam-se ainda mais difíceis de serem resolvidos.

Apesar de haver uma grande literatura disponível para as versões clássicas, trabalhos que consideram a integração do sequenciamento de tarefas junto com custos de entrega dos lotes são menos numerosos. Um dos primeiros trabalhos a abordar este tipo de problema foi realizado por Kahlbacher & Cheng (1993) que resolveu o problema no qual a função objetivo considerava tanto custos de estoques (associados a penalidade de adiantamento) quanto custos de entrega (associados ao número de tarefas atrasadas). Cheng et al. (1996) aborda o problema onde a função objetivo é minimizar a soma do número de lotes entregues e diversas funções de penalidades por adiantamento de tarefas, mas sem considerações de data de liberação. O autor também demonstra a relação deste tipo de problema com problema de máquinas paralelas. Lee & Chen (2001) realizam um trabalho focado na análise de complexidade de diversos problemas com consideração de transporte.

Alguns trabalhos consideram outras funções objetivos que não o fluxo. No trabalho realizado por Mazdeha et al. (2011) é abordada a minimização do número ponderado de tarefas atrasadas, denotada por $1||\sum U_j w_j + \sum d_i\delta_i$. O autor se utiliza da modelagem matemática e de uma heurística *Simulated Annealing* (SA), onde

são mostrados resultados para instâncias com até 42 tarefas. No trabalho realizado por Hamidinia et al. (2012) é abordado o problema que envolve adiantamento, atraso, custo de estoque e custos associados a entrega de cada lote, denotado por $1||\sum \alpha_{ij}T_{ij} + \beta_{ij}E_{ij} + h_{ij}H_{ij} + d_iY_{ik}$. O autor trabalha com duas abordagens. A primeira é através da modelagem matemática, que se mostra inviável devido ao grande número de restrições. A segunda é uma implementação de um algoritmo genético, na qual também é proposto um novo esquema para codificação dos cromossomos. Seus resultados são comparados, através da utilização de 4 instâncias, com uma implementação tradicional de algoritmo genético proposto para o sequenciamento da produção com sistema de entrega que não leva em consideração a formação de lotes. O autor demostra que é possível obter redução de custos ao se considerar o sistema de entrega com formação de lotes, principalmente quando os custos de entregas são altos quando comparados aos custos de estoques. Haddad et al. (2012) investiga o problema de sequenciamento de tarefas em uma máquina com o objetivo de minimizar o atraso máximo (*maximum lateness*) e os custos de entrega dos lotes, simultaneamente. O problema abordado também considera a deterioração no tempo de processamento das tarefas, que depende da posição das mesmas no sequenciamento. O autor propõe, além da implementação de um novo modelo matemático, o desenvolvimento de uma heurística SA e também apresenta um *lower bound* para comparar com a performance do modelo para problemas de grande escala.

O problema $1||\sum F_{ij} + \sum \delta_i d_i$, que não considera data de liberação, é trabalhado por Hall & Potts (2003) que propõe um algoritmo de programação dinâmica para este problema. A versão ponderada deste mesmo problema, denotada por $1||\sum w_{ij}F_{ij} + \sum \delta_i d_i$, foi alvo de estudos por parte de Ji et al. (2007), que provaram que o problema pertence a classe $\mathcal{NP}$-Difícil mesmo quando o número de lotes é constante.

O problema $1|r_{ij}|\sum F_{ij} + \sum \delta_i d_i$, que considera data de liberação, foi trabalhado por Hall & Potts (2003) e por Mahdavi Mazdeh et al. (2008), que resolve o problema, mas com uma pressuposição de que para cada par de tarefas j e j' sempre que $p_j < p_{j'}$ então $r_j < r_{j'}$, através de um algoritmo B&B.

No presente trabalho é abordada a versão ponderada, $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$, que considera tanto a data de liberação, pesos e custos de entrega. No melhor do nosso conhecimento, este problema só havia sido trabalho antes por Mazdeha et al. (2012) que propôs um algoritmo B&B que resolve instâncias com até 40 tarefas, e pressupõe que para cada par de tarefas j e j' sempre que $\frac{p_j}{w_j} \leq \frac{p_{j'}}{w_{j'}}$ então $r_j \leq r_{j'}$. Suposição esta que não é considerada no presente trabalho, de modo que o problema é abordado num contexto mais geral. Selvarajah et al. (2013) também trabalhou com

problema semelhante, $1|r_{ij}|\sum w_{ij}F_{ij} + nd$, porém considera custos fixos na entrega de lotes, propondo *lower bounds* e o desenvolvimento de um algoritmo genético.

Para informações mais completas sobre a literatura disponível de problemas que envolvem a formação de lotes, o leitor pode consultar os *surveys* realizados por Allahverdi et al. (1999), Potts & Kovalyov (2000), Hall & Potts (2003) e Allahverdi et al. (2008).

3.3 Modelo matemático proposto

Um modelo de programação inteira mista para o problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$ é proposto a seguir.

Dados de entrada :

N	número de tarefas
F	número de clientes
J_{ij}	a j-ésima tarefa do cliente i
p_{ij}	tempo de processamento da tarefa J_{ij}
r_{ij}	data de liberação da tarefa J_{ij}
w_{ij}	peso da tarefa J_{ij}
d_i	custo de se entregar um lote ao cliente i
n_i	número de tarefas do cliente i ($n_i \leq N$, $\sum_{i=1}^{F} n_i = N$)
M	número inteiro suficientemente grande

Variáveis de decisão :

C_{ij}	tempo de conclusão da tarefa J_{ij}
F_{ki}	tempo de conclusão do k-ésimo lote do cliente i
θ_{ki}	número de tarefas alocadas ao k-ésimo lote do cliente i

$$X_{ij}^{i'j'} = \begin{cases} 1, & \text{se a tarefa } J_{ij} \text{ precede imediatamente a tarefa } J_{i'j'} \\ 0, & \text{caso contrário} \end{cases}$$

$$Y_{ij}^{k} = \begin{cases} 1, & \text{se a tarefa } J_{ij} \text{ está alocada ao } k\text{-ésimo lote do cliente } i \\ 0, & \text{caso contrário} \end{cases}$$

$$\beta_{ki} = \begin{cases} 1, & \text{se há pelo menos uma tarefa no } k\text{-ésimo lote do cliente } i \\ 0, & \text{caso contrário} \end{cases}$$

3.3.1 Formulação do modelo matemático - não linear

A formulação do modelo é apresentado a seguir:

Função objetivo $$min \quad \sum_{i=1}^{F}\sum_{j=1}^{n_i} w_{ij}\left(\sum_{k=1}^{n_i} Y_{ij}^{k} F_{ki}\right) + \sum_{i=1}^{F}\sum_{k=1}^{n_i} \beta_{ki} d_i \tag{3.1}$$

Sujeito a:

$$\sum_{i=0}^{F}\sum_{j=1}^{n_i} X_{ij}^{i'j'} = 1 \qquad \forall i' = 1 \ldots F, \forall j' = 1 \ldots n_{i'} \tag{3.2}$$

$$\sum_{i'=1}^{F}\sum_{j'=1}^{n_{i'}} X_{ij}^{i'j'} = 1 \qquad \forall i = 0 \ldots F, \forall j = 1 \ldots n_i \tag{3.3}$$

$$C_{i'j'} \geq C_{ij} - M + (M + p_{i'j'}) X_{ij}^{i'j'} \qquad \forall i,i' = 1 \ldots F, \forall j = 1..n_i, \forall j' = 1 \ldots n_{i'} \tag{3.4}$$

$$C_{ij} - p_{ij} \geq r_{ij} \qquad \forall i = 1 \ldots F, \forall j = 1 \ldots n_i \tag{3.5}$$

$$C_{01} = 0 \tag{3.6}$$

$$\theta_{ki} = \sum_{j=1}^{n_i} Y_{ij}^{k} \qquad \forall i = 1 \ldots F, \forall k = 1 \ldots n_i \tag{3.7}$$

$$\theta_{ki} \geq \beta_{ki} \qquad \forall i = 1 \ldots F, \forall k = 1 \ldots n_i \tag{3.8}$$

$$n_i\beta_{ki} \geq \theta_{ki} \qquad \forall i = 1 \dots F, \forall k = 1 \dots n_i \quad (3.9)$$

$$F_{ki} \geq C_{ij} Y_{ij}^k \qquad \forall i = 1 \dots F, \forall j, k = 1 \dots n_i \quad (3.10)$$

$$\sum_{k=1}^{n_i} Y_{ij}^k = 1 \qquad \forall i = 1 \dots F, \forall j = 1 \dots n_i \quad (3.11)$$

$$X_{ij}^{i'j'} \in \{0, 1\} \qquad \forall i,i' = 0 \dots F, \forall j = 1 \dots n_i, \forall j' = 1 \dots n_{i'} \quad (3.12)$$

$$Y_{ij}^k \in \{0, 1\} \qquad \forall i = 1 \dots F, \forall j,k = 1 \dots n_i \quad (3.13)$$

$$\beta_{ki} \in \{0, 1\} \qquad \forall i = 1 \dots F, \forall k = 1 \dots n_i \quad (3.14)$$

$$C_{ij} \geq 0 \qquad \forall i = 0 \dots F, \forall j = 1 \dots n_i \quad (3.15)$$

$$F_{ki} \geq 0 \qquad \forall i = 1 \dots F, \forall k = 1 \dots n_i \quad (3.16)$$

$$\theta_{ki} \geq 0 \qquad \forall i = 1 \ldots F, \forall k = 1 \ldots n_i \quad (3.17)$$

Para a construção deste modelo considera-se inicialmente um conjunto de lotes vazios, específicos para cada um dos clientes, de modo que, para um cliente i ($\forall i = 1 \ldots F$), haja inicialmente n_i lotes vazios. Cada tarefa deste cliente é, então, alocada a um destes lotes. Possivelmente, mais de uma tarefa seja alocada a um mesmo lote e alguns lotes fiquem vazios. Neste último caso, os lotes vazios são desconsiderados no cálculo da função objetivo. Ao considerar que cada cliente possui um conjunto específico de lotes aos quais suas tarefas são alocadas, já se evita que uma tarefa de um cliente esteja em um lote de outro cliente, o que também é uma restrição importante do problema. Para o desenvolvimento deste modelo foi adotada a estratégia de dividir o problema em duas partes. A primeira, modelada pelas restrições (3.2)-(3.6), resolve o problema de sequenciamento com data de liberação de tarefas clássico. A outra parte, modelada pelas restrições (3.7)-(3.11), decide a alocação de tarefas aos lotes. Assume-se também que há um cliente fictício 0 que possui uma única tarefa, que é a tarefa fictícia J_{01} e que possui tempo de processamento igual a 0 e custo de entrega $d_0 = 0$. Neste modelo, a tarefa fictícia J_{01} sempre antecede a primeira tarefa do sequenciamento e é utilizada apenas para efeito do cálculo do tempo de conclusão da primeira tarefa do sequenciamento. Esta parte do modelo teve como motivação os modelos apresentados nos trabalhos Mazdeha et al. (2011) e Haddad et al. (2012).

A equação (3.1) introduz a função objetivo, onde o primeiro termo calcula o fluxo ponderado total e o segundo o custo total de entrega. Para calcular o fluxo ponderado da tarefa J_{ij}, basta multiplicar o seu peso, w_{ij}, pelo tempo de conclusão efetivo desta tarefa que é exatamente o mesmo tempo de conclusão do lote a que esta tarefa está alocada: $\sum_{k=1}^{n_i} Y_{ij}^k F_{ki}$.

As restrições (3.2) e (3.3) afirmam, respectivamente, que cada tarefa possui exatamente uma tarefa que a antecede e exatamente uma tarefa que a sucede. As restrições (3.4) calculam o tempo de conclusão de cada tarefa. Se $X_{ij}^{i'j'} = 1$, significa que a tarefa J_{ij} precede imediatamente a tarefa $J_{i'j'}$. Logo, o tempo de conclusão da tarefa $J_{i'j'}$ será dado por $C_{i'j'} \geq C_{ij} + p_{i'j'}$. Caso $X_{ij}^{i'j'} = 0$, a restrição se torna redundante. As restrições (3.5) definem que nenhuma tarefa deva começar o seu processamento antes da sua data de liberação. A restrição (3.6) define o tempo de conclusão da primeira tarefa do cliente fictício 0 como sendo zero.

As restrições (3.7) calculam o número de tarefas para cada um dos lotes de cada cliente. As restrições (3.8) e (3.9) fazem a associação entre as variáveis θ e β. As variáveis β_{ki} assumem valor 1 se o k-ésimo lote do cliente i for utilizado ($\theta_{ki} \geq 1$) e assumem o valor 0 caso o lote não seja utilizado ($\theta_{ki} = 0$). As restrições (3.10) são responsáveis por determinar o tempo de conclusão do k-ésimo lote do cliente i, que é o maior tempo de conclusão dentre as tarefas do cliente i que estão neste lote. As restrições (3.11) determinam que cada tarefa deve estar em exatamente um lote do seu cliente.

As restrições (3.12), (3.13) e (3.14) mostram, respectivamente, que as variáveis X, Y e β são variáveis binárias. Finalmente, as restrições (3.15), (3.16) e (3.17) afirmam, respectivamente, que C, F e θ são variáveis que assumem valores não-negativos.

3.3.2 Formulação do modelo matemático - linearizado

Observe que no modelo apresentado na Seção 3.3.1, as restrições (3.10) e a função objetivo (3.1) são não-lineares, por causa da multiplicação entre as variáveis de decisão C e Y e entre Y e F, respectivamente. Para linearizar o modelo foram realizadas as seguintes substituições:

- $\Delta_{ij}^{k} = C_{ij} Y_{ij}^{k}$
- $\alpha_{ij}^{k} = Y_{ij}^{k} F_{ki}$

3.3.2.1 Substituição Δ_{ij}^{k}

Esta substituição deve ser feita de tal forma que:

$$\begin{cases} Y_{ij}^{k} = 0 \Rightarrow \Delta_{ij}^{k} = 0 \\ Y_{ij}^{k} = 1 \Rightarrow \Delta_{ij}^{k} = C_{ij} \end{cases}$$

Com isso, as restrições (3.10) serão substituídas por (3.18) e será necessário adicionar as restrições (3.19), (3.20), (3.21) e (3.22) ao modelo.

$$F_{ki} \geq \Delta_{ij}^{k} \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.18)$$

$$\Delta_{ij}^{k} \leq C_{ij} \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.19)$$

$$\Delta_{ij}^k \geq C_{ij} - M(1 - Y_{ij}^k) \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.20)$$

$$\Delta_{ij}^k \leq M Y_{ij}^k \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.21)$$

$$\Delta_{ij}^k \geq 0 \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.22)$$

Explicação:	Explicação:
Se $Y_{ij}^k = 0$ então:	Se $Y_{ij}^k = 1$ então:
• $\Delta_{ij}^k \leq C_{ij}$ de (3.19)	• $\Delta_{ij}^k \leq C_{ij}$ de (3.19)
• $\Delta_{ij}^k \geq C_{ij} - M$ de (3.20) - redundante	• $\Delta_{ij}^k \geq C_{ij}$ de (3.20)
• $\Delta_{ij}^k \leq 0$ de (3.21)	• $\Delta_{ij}^k \leq M$ de (3.21) - redundante
• $\Delta_{ij}^k \geq 0$ de (3.22)	• $\Delta_{ij}^k \geq 0$ de (3.22)
Logo, $\Delta_{ij}^k = 0$	Logo, $\Delta_{ij}^k = C_{ij}$

3.3.2.2 Substituição α_{ij}^k

Esta substituição deve ser feita de tal forma que:

$$\begin{cases} Y_{ij}^k = 0 \Rightarrow \alpha_{ij}^k = 0 \\ Y_{ij}^k = 1 \Rightarrow \alpha_{ij}^k = F_{ki} \end{cases}$$

Com esta substituição a função objetivo (3.1) será substituída por (3.23) e será necessário adicionar as restrições (3.24), (3.25), (3.26) e (3.27) ao modelo:

$$\text{Função objetivo} \qquad min \quad \sum_{i=1}^{F} \sum_{j=1}^{n_i} w_{ij} \left(\sum_{k=1}^{n_i} \alpha_{ij}^k \right) + \sum_{i=1}^{F} \sum_{k=1}^{n_i} \beta_{ki} d_i \quad (3.23)$$

$$\alpha_{ij}^k \leq F_{ki} \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.24)$$

$$\alpha_{ij}^k \geq F_{ki} - M(1 - Y_{ij}^k) \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.25)$$

$$\alpha_{ij}^{k} \leq MY_{ij}^{k} \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.26)$$

$$\alpha_{ij}^{k} \geq 0 \qquad \forall i = 1 \ldots F, \forall j,k = 1 \ldots n_i \quad (3.27)$$

Explicação:	Explicação:
Se $Y_{ij}^{k} = 0$ então:	Se $Y_{ij}^{k} = 1$ então:
• $\alpha_{ij}^{k} \leq F_{ki}$ de (3.24)	• $\alpha_{ij}^{k} \leq F_{ki}$ de (3.24)
• $\alpha_{ij}^{k} \geq F_{ki} - M$ de (3.25) - redundante	• $\alpha_{ij}^{k} \geq F_{ki}$ de (3.25)
• $\alpha_{ij}^{k} \leq 0$ de (3.26)	• $\alpha_{ij}^{k} \leq M$ de (3.26) - redundante
• $\alpha_{ij}^{k} \geq 0$ de (3.27)	• $\alpha_{ij}^{k} \geq 0$ de (3.27)
Logo, $\alpha_{ij}^{k} = 0$	Logo, $\alpha_{ij}^{k} = F_{ki}$

As restrições (3.27) e (3.22) afirmam, respectivamente, que α e Δ são variáveis que assumem valores não-negativos. De modo geral, o número de variáveis e restrições utilizadas no modelo linearizado são $N^2 + 3\sum_{i=1}^{F} {n_i}^2 + 6N + 2$ variáveis, sendo $N^2 + \sum_{i=1}^{F} {n_i}^2 + 3N + 1$ variáveis inteiras binárias, e $2N^2 + 10\sum_{i=1}^{F} {n_i}^2 + 13N + 5$ restrições.

3.3.3 Formulação do modelo matemático - remoção de simetria

É possível observar que algumas variáveis neste modelo são simétricas, notadamente as variáveis Y_{ij}^{k} e β_{ki}, que definem a alocação de uma determinada tarefa J_{ij} a um lote de índice k do cliente i.

A situação de simetria mencionada anteriormente está exemplificada na Figura 3.1 onde são mostradas duas soluções equivalentes quanto a formação de lotes para um determinado cliente a, que possui 4 tarefas: J_{a1}, J_{a2}, J_{a3}, J_{a4}. As tarefas J_{a1} e J_{a2} estão em um lote e as tarefas J_{a3} e J_{a4} em outro. As soluções 1 e 2, mostradas na Figura 3.1, são equivalentes, de modo que não é necessário considerar todas as possibilidade de formação de lotes e, assim, a solução 2 e outras simétricas são eliminadas.

Neste modelo, sem perda de generalidade, somente são consideradas as formações de lotes que ocupem primeiro os lotes de menor índice. Para um cliente i qualquer ($\forall i = 1 \ldots F$), um lote k' ($\forall k' = 2 \ldots n_i$), somente será utilizado se os

Solução 1:	Solução 2:
• Lote 1 $\rightarrow \{J_{a1},\ J_{a2}\}$	• Lote 1 $\rightarrow \{J_{a1},\ J_{a2}\}$
• Lote 2 $\rightarrow \{J_{a3},\ J_{a4}\}$	• Lote 2 $\rightarrow \emptyset$
• Lote 3 $\rightarrow \emptyset$	• Lote 3 $\rightarrow \{J_{a3},\ J_{a4}\}$
• Lote 4 $\rightarrow \emptyset$	• Lote 4 $\rightarrow \emptyset$

Figura 3.1. Exemplo de ocorrência de simetria

lotes anteriores a ele já tiverem sido utilizados para alguma tarefa, como ocorre na solução 1 da Figura 3.1. Por sua vez, a solução 2 não será considerada já que o lote de índice 3 fora utilizado e o lote com índice inferior (no caso, o lote 2) não. As restrições (3.28) removem a simetria que ocorre com respeito às variáveis β_{ki} enquanto que as restrições (3.29) estão responsáveis pela remoção de simetria associadas às variáveis Y_{ij}^k.

$$\beta_{ki} \geq \beta_{k'i} \qquad \forall i = 1 \ldots F, \forall k = 1 \ldots n_i - 1, \forall k' = k+1 \ldots n_i \quad (3.28)$$

$$\sum_{j'=1}^{n_i} \sum_{k'=1}^{k-1} Y_{i'j'}^{k'} - Y_{ij}^k \geq 0 \qquad \forall i = 1 \ldots F, \forall j = 1 \ldots n_i, \forall k = 2 \ldots n_i \quad (3.29)$$

Observe que, no conjunto de restrições (3.29), se $Y_{ij}^k = 0$ (significando que a tarefa J_{ij} não está alocada ao k-ésimo lote do cliente i) os lotes $k' < k$ podem ter sido utilizados ou não: $\sum_{j'=1}^{n_i} \sum_{k'=1}^{k-1} Y_{i'j'}^{k'} \geq 0$, tornando as restrições (3.29) redundantes. Por outro lado, se $Y_{ij}^k = 1$ então os lotes de índice $k' < k$ devem ter sido utilizados ou, em outras palavras, não deve existir lote $k' < k$ que não foi utilizado. A única situação em que as restrições (3.29) tornam-se inválidas ocorre quando $Y_{ij}^k = 1$ (um lote $k > 2$ foi utilizado) e o somatório $\sum_{j'=1}^{n_i} \sum_{k'=1}^{k-1} Y_{i'j'}^{k'} = 0$ (os lotes $k' < k$ não foram utilizados).

3.4 Heurísticas propostas

Nesta seção estão descritas as heurísticas propostas. Optou-se inicialmente por propor uma heurística baseada na metaheurística *Iterated Local Search* (ILS), que é uma heurística de simples implementação e bastante eficiente na solução de problemas de sequenciamento de tarefas (Lourenço et al., 2003). No melhor do nosso conhecimento, a metaheurística ILS não foi aplicada para este problema. A

outra heurística proposta é baseada na metaheurística *Iterated Greedy* (IG) que tem obtidos excelentes resultados para diversos problemas de sequenciamento (Ruiz & Stützle, 2007). Tanto ILS quanto IG utilizam a mesma representação de solução, a mesma heurística construtiva e o mesmo procedimento de busca local que são descritos, respectivamente, nas Seções 3.4.1, 3.4.2 e 3.4.3. Os componentes específicos do ILS e IG são descritos na Seções 3.4.4 e 3.4.5, respectivamente.

3.4.1 Representação de uma solução

Para representar uma solução S para este problema devem ser consideradas tanto a sequência das tarefas quanto a alocação de tarefas aos lotes. A sequência de tarefas é representada por uma permutação S_J de tamanho N: $S_J = \{j_1, j_2, \ldots, j_N\}$, onde j_k é a k-ésima tarefa na sequência de processamento, $\forall k = 1,2,\ldots,N$. A alocação de tarefas aos lotes é feita através de um arranjo S_B de tamanho N, onde em cada posição k de S_B, $\forall k = 1,2,\ldots,N$, é armazenado um número inteiro que representa o lote, do respectivo cliente, ao qual a tarefa j_k está alocada. Se a tarefa j_k é para ser entregue ao cliente i, então $1 \leq S_B[k] \leq n_i$. Observe que o limitante superior foi definido como n_i para que se tenha no máximo n_i lotes para o cliente i. Esta situação ocorre quando todas as tarefas do cliente i são entregues separadas. Então, uma solução é definida por $S = (S_J, S_B)$.

Para efeito de ilustração, considere a instância especificada na Tabela 3.1, onde os valores do tempo de processamento, do peso e da data de liberação estão especificados por p_{ij}, w_{ij} e r_{ij}, respectivamente. Considere também a solução S mostrada na Figura 3.2, que é a solução ótima para esta instância. Cada lote será denotado por B_{ik}, sendo entendido como o k-ésimo lote do cliente i. Cada lote B_{ik} corresponde a um conjunto contendo as tarefas do cliente i que serão entregues juntas no lote k. Pode-se ver na Figura 3.2 que há formação de três lotes: lotes $B_{11} = \{J_{11}, J_{12}\}$ e $B_{12} = \{J_{13}\}$ do cliente 1; e lote $B_{21} = \{J_{21}, J_{22}\}$ do cliente 2. Essa formação de lotes significa que as tarefas J_{11} e J_{12} serão entregues juntas para o cliente 1 no tempo 70, acarretando um custo de entrega, que no caso é de 2000. O lote B_{12} é considerado ser um lote com uma única tarefa, que no caso é a tarefa J_{13} acarretando mais um custo de entrega de 2000 ao cliente 1. Já o lote B_{21} contém as tarefas J_{21} e J_{22}, que serão entregues juntas no tempo 140 com um único custo de entrega de 1800 associado a entrega do lote para o cliente 2.

Ainda considerando o exemplo numérico da Tabela 3.1 e a solução mostrada na Figura 3.2, tem-se a sequência de processamento das tarefas $S_J = \{J_{11}, J_{21}, J_{12}, J_{22}, J_{13},\}$ e o arranjo de lotes $S_B = \{1, 1, 1, 1, 2\}$. Observe que nas

duas primeiras posições do arranho S_B é armazenado o mesmo valor 1. Este valor não quer dizer que as tarefas J_{11} e J_{21} estejam no mesmo lote (até porque são tarefas que pertencem a clientes diferentes). Quer dizer que a tarefa J_{11} está no lote 1 do seu respectivo cliente (o lote B_{11} do cliente 1) e que a tarefa J_{21} está no lote 1 do cliente 2 (o lote B_{21}). O fluxo ponderado total (TWFT) será: TWFT $= 70 * (5 + 6) + 140 * (5 + 10) + 290 * 18 = 8090$. O custo total de entrega (TDC) será: TDC $= 2 * 2000 + 1 * 1800 = 5800$. Logo, o valor da função objetivo para esta solução S, denotado por $f(S)$, será $f(S) = \text{TWFT} + \text{TDC} = 8090 + 5800 = 13890$.

Tabela 3.1. Dados de entrada para uma instância com 5 tarefas e 2 clientes

	Cliente 1			**Cliente 2**	
Tarefa J_{ij}	J_{11}	J_{12}	J_{13}	J_{21}	J_{22}
p_{ij}	15	30	150	20	70
w_{ij}	5	6	18	5	10
r_{ij}	0	40	80	10	70

(a) Tempo de processamento, peso e data de liberação

Clientes	1	2
$\mathbf{d_i}$	2000	1800

(b) Custos de entrega para os cliente 1 e 2

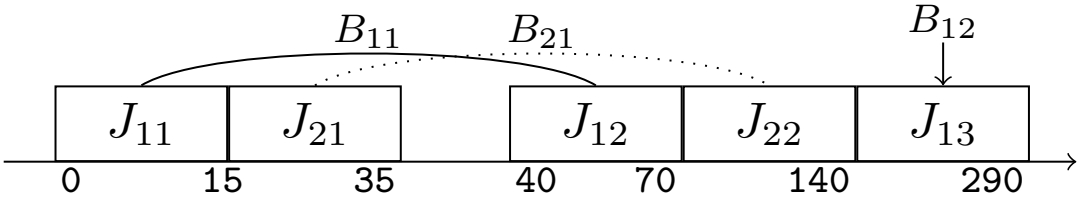


Figura 3.2. Solução $S = (S_J, S_B)$ definida pelo sequenciamento de tarefas $S_J = \{J_{11}, J_{21}, J_{12}, J_{22}, J_{13}\}$ e arranjo de lotes $S_B = \{1, 1, 1, 1, 2\}$. Nesta solução existem três lotes formados: $B_{11} = \{J_{11}, J_{12}\}$, $B_{12} = \{J_{13}\}$ e $B_{21} = \{J_{21}, J_{22}\}$.

3.4.2 Construção da solução inicial

Para construir uma solução inicial foi proposta uma heurística baseada na regra de despacho WSPT - *Weighted Shortest Processing Time*. Nesta regra, as tarefas são ordenadas de forma não-decrescente pelo tempo de processamento ponderado que é definido pela razão p_{ij}/w_{ij}. Desta forma é inicializada a permutação S_J.

Com respeito ao arranjo S_B, para alocação das tarefas aos lotes, considera-se que somente 1 lote será entregue para cada cliente, ou seja, todas as tarefas do mesmo cliente estão no mesmo lote: $S_B[k] = 1, \forall k = 1, \ldots, N$. Assim, tem-se uma solução inicial viável para o problema.

Observe que enquanto a primeira parte da geração da solução inicial (regra WSPT) procura tratar adequadamente o fluxo ponderado total a segunda parte procura tratar adequadamente os custos de entrega (o menor custo de entrega será quando houver uma única entrega para cada cliente). No Algoritmo 5 está descrito o pseudocódigo da heurística construtiva utilizada. Na linha 2, todas as tarefas serão ordenadas de acordo com a regra WSPT. Em seguida, nas linhas 3 e 4, é construído um único lote para cada cliente e todas as tarefas do respectivo cliente são alocadas ao mesmo lote. Ao final é gerada uma solução inicial viável S que é composta pela permutação de tarefas S_J e pelo arranjo de lotes S_B (linha 5).

Algoritmo 5: CONSTRUCTION ()

```
   output: A feasible solution S
1  begin
2  |  S_J := WSPT(); // Short jobs by WSPT and iniciate the
   |  permutation S_J
3  |  for k := 1 to N do
4  |  |  S_B[k] := 1;
5  |  S := (S_J, S_B) ; // Compose a solution S with permutation S_J
   |  and arrange S_B
6  |  return S
```

3.4.3 Busca local

A busca local é usada para melhorar a solução inicial S e as soluções obtidas pelo procedimento de perturbação. Busca local é um método que começa com uma solução S e gera uma vizinhança que contém soluções que são alcançadas através de movimentos individuais realizados na solução corrente. Desta vizinhança, uma solução que é melhor que a solução corrente é selecionada. A solução escolhida torna-se a nova solução corrente e o processo continua até que um ótimo local seja alcançado.

O procedimento de busca local utilizado gera uma vizinhança baseada em movimentos de trocas de pares de tarefas adjacentes e outra vizinhança geradas através de trocas de pares aleatórios. O procedimento de busca local será denominado de LOCAL_SEARCH e o seu pseudocódigo está descrito no Algoritmo 6.

O processo iterativo da busca local ocorre nas linhas 3-9 enquanto houver melhora na solução corrente S. Em cada uma das iterações são aplicados em sequência dois outros algoritmos de busca local. Primeiro, é aplicado o procedimento API_LS (linha 5) sobre a solução corrente S gerando um ótimo local S_1. A seguir, é gerado outro ótimo local S_2 a partir de S_1 pelo método RPI_LS (linha 6). A solução S_2 pode substituir a solução corrente S caso seja melhor. O método LOCAL_SEARCH possui dois parâmetros de entrada: uma solução S que é a solução alvo que será melhorada e *numberSwap* que é o número máximo de trocas aleatórias a serem feitas no procedimento RPI_LS (observe a linha 6).

Algoritmo 6: LOCAL_SEARCH (S, *numberSwap*)

```
   output: Improved Solution S
 1 begin
 2 |  improve := true;
 3 |  while improve do
 4 |  |  improve := false;
 5 |  |  S1 := API_LS (S);
 6 |  |  S2 := RPI_LS (S1, numberSwap);
 7 |  |  if f(S2) < f(S) then
 8 |  |  |  improve := true;
 9 |  |  |  S := S2;
10 |  return S
```

O procedimento API_LS (*Adjacent Pairwise Interchange Local Search*) é mostrado no Algoritmo 7. Ele é um procedimento de busca local baseado em vizinhança de trocas de pares adjacentes. Daqui em diante, para diferenciar a permutação de tarefas (S_J) e o arranjo de lotes (S_B) de duas ou mais soluções quaisquer, será utilizada a seguinte notação: $S_X.S_J$ para representar a permutação de tarefas S_J de uma solução S_X e $S_X.S_B$ para representar o arranjo de lotes S_B de uma solução S_X.

No Algoritmo 7, começando da primeira posição ($k = 1$) até a penúltima posição do sequenciamento ($N-1$), veja linhas 2 e 3, é realizado a troca das tarefas, e dos respectivos lotes, nas posições k e $k+1$ (linhas 4 e 5). Se não houve melhora, a troca é desfeita e o próximo par da solução é considerado. Caso haja melhora, a solução corrente é atualizada e o procedimento recomeça da primeira posição (linhas 6 a 8). Observe que é utilizada a estratégia de primeiro aprimorante na busca local, já que a busca local é reiniciada assim que se encontra um primeiro vizinho S'_1 melhor que S (linha 6-8).

O outro procedimento de busca local utilizado por LOCAL_SEARCH é o método RPI_LS (*Randomized Pairwise Interchange Local Search*) que é mostrado

Algoritmo 7: API_LS (S)

```
   output: Improved Solution S
1  begin
2      k := 1;
3      while k ≤ N − 1 do
4          S_1 := SWAP (S.S_J[k], S.S_J[k + 1]); // Swap jobs.  Not modify S
5          S'_1 := SWAP (S_1.S_B[k], S_1.S_B[k + 1]); // Swap batches
6          if f(S'_1) < f(S) then
7              S := S'_1; // Accept the swap
8              k := 1; // The process is reinitiated
9      return S
```

no Algoritmo 8. Ele é um procedimento de busca local baseado em vizinhança de trocas aleatórias entre tarefas. O procedimento possui o parâmetro *numberSwap*, que define o número de trocas a serem realizadas. Neste procedimento, são sorteadas duas posições diferentes, pos_1 e pos_2 (linhas 3 e 4), e, então, é realizada a troca das tarefas, e dos respectivos lotes, que estão nas posições pos_1 e pos_2 (linhas 5 e 6). Se houver melhora, a solução corrente é atualizada (linhas 7 e 8). É importante observar que esta busca local não avaliar de forma exaustiva a vizinhança de S.

Algoritmo 8: RPI_LS (S, *numberSwap*)

```
   output: Improved Solution S
1  begin
2      for i := 1 to numberSwap do
3          pos_1 := RAND (1..N);
4          pos_2 := RAND (1..N); // pos_2 ≠ pos_1
5          S_1 := SWAP (S.S_J[pos_1], S.S_J[pos_2]); // Swap jobs.  Not modify S
6          S'_1 := SWAP (S_1.S_B[pos_1], S_1.S_B[pos_2]); // Swap Batches
7          if f(S'_1) < f(S) then
8              S := S'_1; // Accept the swap
9      return S
```

3.4.3.1 Regras de dominância

Neste trabalho são propostas seis regras de dominância para redução da vizinhança de troca de pares adjacentes. Estas regras identificam soluções que são dominadas por outras, de modo que seja possível prever se a função objetivo de uma solução S irá melhorar, piorar ou permanecer inalterada quando se utiliza a vizinhança de troca de pares adjacentes, que é o tipo de vizinhança utilizada na busca local API_LS. A definição destas regras é importante para que não seja preciso calcular a função

objetivo de uma solução dominada. O cálculo da função objetivo normalmente é uma das partes que mais requerem processamento em uma metaheurística. Logo, evitar a geração de alguma soluções vizinhas, quando possível, acelera a execução da heurística realizando um maior número de iterações, num mesmo intervalo de tempo, obtendo melhores resultados.

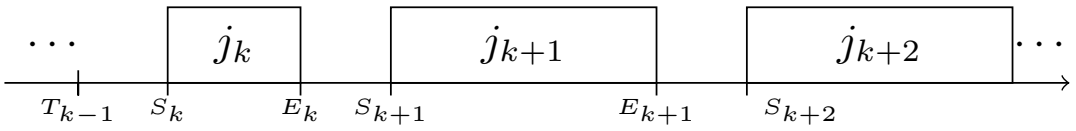
Para explicar como foi realizada a redução da vizinhança na busca local, considere duas tarefas consecutivas nas posições k e $k+1$ de um determinado sequenciamento de tarefas S_J de uma solução S. Estas tarefas simplesmente serão denotadas por j_k e j_{k+1} e os lotes aos quais as tarefas j_k e j_{k+1} estão alocadas serão designados por b_k e b_{k+1}, respectivamente. Para estas duas tarefas deseja-se determinar se vale a pena ou não realizar a troca. Seja T_{k-1} o tempo em que a máquina está disponível para processar j_k. E seja j_{k+2} a tarefa seguinte à tarefa j_{k+1}. Se j_{k+1} é a última tarefa da sequência, então, j_{k+2} é uma tarefa fictícia que inicia o seu processamento imediatamente após j_{k+1} terminar de ser processada.

Além disso, antes de realizar a troca das duas tarefas, define-se o tempo de início do processamento de j_k sendo S_k e o fim do processamento desta tarefa sendo E_k. Após a troca, os tempos de início e fim do processamento de j_k serão S'_k e E'_k, respectivamente. Estas mesmas variáveis são definidas de forma análoga para a tarefa j_{k+1}: S_{k+1}, E_{k+1}, S'_{k+1} e E'_{k+1}. A Figura 3.3 ilustra o que foi dito anteriormente. Na Figura 3.3(a) está representado o sequenciamento de tarefas S_J antes da troca e na Figura 3.3(b) está o sequenciamento de tarefas S'_J após a troca.

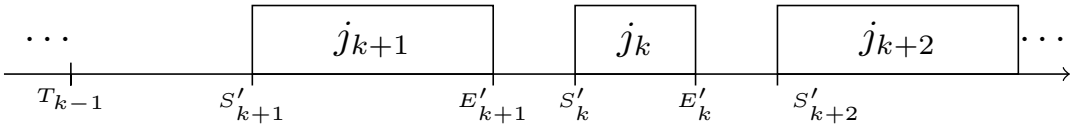
Dada uma solução S sobre a qual irá se aplicar o algoritmo API_LS é sabido à *priori* os valores de S_k, E_k, S_{k+1} e E_{k+1} e também o tempo de conclusão dos lotes b_k e b_{k+1}, que é o tempo de interesse no cálculo do fluxo ponderado, sendo denominados, respectivamente, simplesmente por F_k e F_{k+1}. Denota-se por W_k e W_{k+1} o peso total dos lotes b_k e b_{k+1}, respectivamente. Após a realização da troca das tarefas j_k e j_{k+1} tem-se que:

- $S'_{k+1} = max(T_{k-1}, r_{k+1})$
- $E'_{k+1} = S'_{k+1} + p_{k+1}$
- $S'_k = max(E'_{k+1}, r_k)$
- $E'_k = S'_k + p_k$

Observe que neste tipo de busca local, o tempo de início e fim de processamento das tarefas anteriores a tarefa j_k em S_J (ou anteriores a j_{k+1} em S'_J) não se alteraram. Observe também que a formação de lotes não é alterada, de modo que, os custos de



(a) Sequenciamento de tarefas S_J antes da troca.



(b) Sequenciamento de tarefas S'_J depois da troca.

Figura 3.3. Sequenciamento de tarefas S_J antes e depois de trocar as tarefas j_k e j_{k+1}. Foi omitida nesta representação a alocação de tarefas aos lotes.

entrega sejam os mesmos antes e depois da troca. Assim, para saber se a solução S' após a troca é pior que a solução original S basta analisar o tempo de conclusão das tarefas subsequentes a j_k em S_J (ou subsequentes a j_{k+1} em S'_J).

Proposição 1 *Numa solução S, sejam j_k e j_{k+1}, $\forall k = 1,2,\ldots,N-1$, duas tarefas* ***consecutivas****, do mesmo cliente i, e que estão associadas ao mesmo lote. Se $E'_k \geq E_{k+1}$, então, a solução S', obtida pela troca das tarefas j_k e j_{k+1}, será dominada pela solução S. Ou seja, S' não será melhor que S.*

Prova:

1. Como as tarefas estão no mesmo lote, então, $b_k = b_{k+1}$ e $F_k = F_{k+1}$ e $F_{k+1} \geq E_{k+1}$
2. Após a troca, $F'_k = F'_{k+1}$ e $F'_k \geq E'_k$
3. Se $E'_k \geq E_{k+1}$ então $F'_k \geq E'_k \geq E_{k+1}$

Note que não há melhoria no tempo de conclusão dos lotes b_k (e b_{k+1}), pois
$\begin{cases} F_{k+1} \geq E_{k+1} \\ F'_k \geq E_{k+1} \end{cases}$

É fácil mostrar que as tarefas posteriores a j_{k+1}, em S_J, também não terão o tempo de conclusão melhorado. Sabe-se que:

1. $S_{k+2} \geq E_{k+1}$
2. $S'_{k+2} \geq E'_k$

3. Se $E'_k \geq E_{k+1}$ então $S'_{k+2} \geq E_{k+1}$. Em outras palavras significa que j_{k+2} não começa em um tempo inferior àquele antes da troca de modo que as tarefas subsequentes a j_{k+2} também não terão o seu tempo de início e fim melhorados.

Mostrou-se que se $E'_k \geq E_{k+1}$ as tarefas posteriores j_{k+1}, em S_J, não terão o tempo de conclusão melhorados e que o fluxo do lote b_k e b_{k+1} também não melhora. Logo o fluxo ponderado total também não irá melhorar □.

Proposição 2 *Numa solução S, sejam j_k e j_{k+1}, $\forall k = 1,2,\ldots,N-1$, duas tarefas* ***consecutivas*** *na sequência de tarefas de S. Se $E'_k = S_{k+2}$; ou se $E'_k < S_{k+2}$ e $S_{k+2} = r_{k+2}$, então, na solução S', obtida pela troca das tarefas j_k e j_{k+1}, o tempo de conclusão das tarefas posteriores a j_k, em S'_J, serão os mesmos que eram em S_J.*

Prova:

A primeira situação, onde $E'_k = S_{k+2}$, é mostrada na Figura 3.4. Sabe-se que $S'_{k+2} \geq E'_k$. Logo, se $E'_k = S_{k+2}$ ocorre, então $S'_{k+2} \geq S_{k+2}$. Ou seja, após a troca a máquina está disponível para processar j_{k+2} a partir de $S'_{k+2} = S_{k+2}$ □.

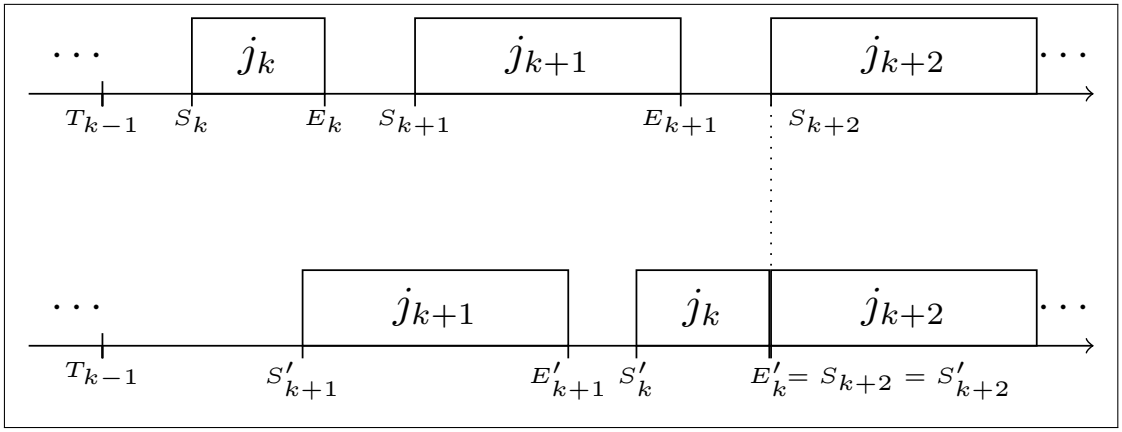


Figura 3.4. Sequenciamento de tarefas antes e depois de trocar as tarefas j_k e j_{k+1}. Caso particular onde após a troca $E'_k = S_{k+2}$.

A outra situação, onde $E'_k < S_{k+2}$ e $S_{k+2} = r_{k+2}$, é mostrada na Figura 3.5. Sabe-se que após a troca, $S'_{k+2} = max(E'_k, r_{k+2})$. Logo, se $r_{k+2} = S_{k+2}$, então $S'_{k+2} = max(E'_k, S_{k+2})$. Se $E'_k < S_{k+2}$ então $S'_{k+2} = S_{k+2}$. Ou seja, após a troca a máquina está disponível para processar j_{k+2} a partir de $S'_{k+2} = S_{k+2}$ □.

Proposição 3 *Numa solução S, sejam j_k e j_{k+1}, $\forall k = 1,2,\ldots,N-1$, duas tarefas* ***consecutivas*** *na sequência de tarefas de S. Seja $\Delta = W_k(F'_k - F_k) + W_{k+1}(F'_{k+1} - F_{k+1})$. Se $E'_k = S_{k+2}$; ou se $E'_k < S_{k+2}$ e $r_{k+2} = S_{k+2}$, então, na solução S', obtida*

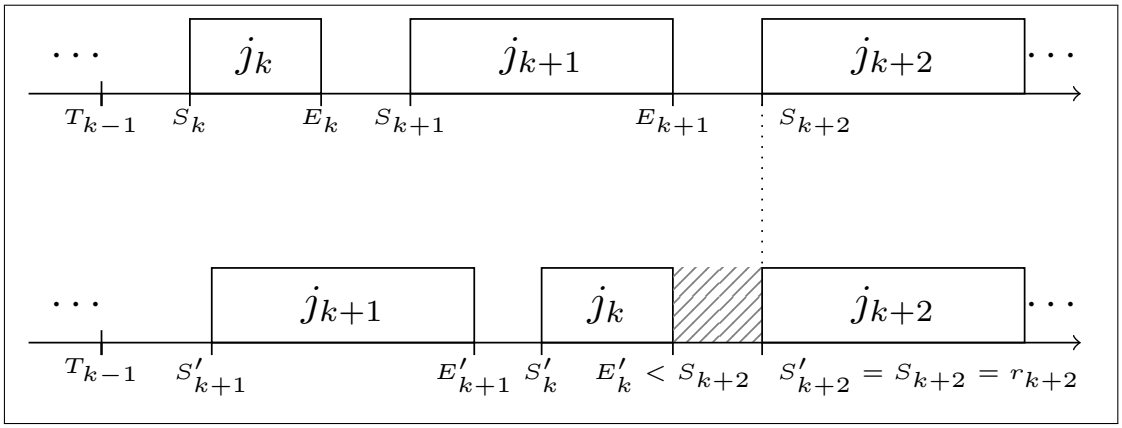


Figura 3.5. Sequenciamento de tarefas antes e depois de trocar as tarefas j_k e j_{k+1}. Caso particular onde após a troca $E'_k < S_{k+2}$ e $S_{k+2} = r_{k+2}$.

pela troca das tarefas j_k e j_{k+1}, o valor da função objetivo será incrementado ou decrementado por Δ. Logo, se $\Delta < 0$, a solução S' dominará S.

Prova:

Como demonstrado na Proposição 2, se $E'_k = S_{k+2}$ ou $E'_k < S_{k+2}$ e $r_{k+2} = S_{k+2}$ então o tempo de conclusão das tarefas posteriores a j_k, em S'_J, serão os mesmos que eram em S_J. O tempo de conclusão das tarefas anteriores a j_{k+1}, em S'_J, também serão os mesmos que eram em S_J de modo que a alteração na função objetivo será possível somente pelos tempos de conclusões das tarefas j_k e j_{k+1} e consequentemente dos respectivos tempos de conclusão, F'_k e F'_{k+1}, dos lotes b_k e b_{k+1}. Com respeito à tarefa j_k deve-se distinguir dois casos:

- $E_k = F_k$. Neste caso, j_k era a última tarefa a ser processada do lote b_k e após a troca continuará sendo a última
- $E_k \neq F_k$. Neste caso, j_k não era a última tarefa a ser processada do lote b_k e após a troca continuará não sendo a última

E também para tarefa j_{k+1}:

- $E_{k+1} = F_{k+1}$. Neste caso, j_{k+1} era a última tarefa a ser processada do lote b_{k+1} e após a troca continuará sendo a última
- $E_{k+1} \neq F_{k+1}$. Neste caso, j_{k+1} não era a última tarefa a ser processada do lote b_{k+1} e após a troca continuará não sendo a última

Como o tempo de conclusão das tarefas posteriores a j_k, em S'_J, não se alteram, pode-se determinar F'_k e F'_{k+1}:

$$F'_k = \begin{cases} F_k, \text{ se } E_k \neq F_k \\ E'_k, \text{ se } E_k = F_k \end{cases} \quad \text{e} \quad F'_{k+1} = \begin{cases} F_{k+1}, \text{ se } E_{k+1} \neq F_{k+1} \\ E'_{k+1}, \text{ se } E_{k+1} = F_{k+1} \end{cases}$$

Antes da troca, os custos associados aos lotes b_k e b_{k+1}, com respeito ao fluxo ponderado total, eram $W_k F_k$ e $W_{k+1} F_{k+1}$, respectivamente. Após a troca de j_k e j_{k+1}, as parcelas serão $W_k F'_k$ e $W_{k+1} F'_{k+1}$, respectivamente. Logo, o valor da função objetivo será alterado por:

$$\Delta = W_k F'_k - W_k F_k + W_{k+1} F'_{k+1} - W_{k+1} F_{k+1}$$
$$\Delta = W_k (F'_k - F_k) + W_{k+1} (F'_{k+1} - F_{k+1}) \square$$

Proposição 4 *Numa solução S, sejam j_k e j_{k+1}, $\forall k = 1,2,\ldots,N-1$, duas tarefas* ***consecutivas*** *na sequência de tarefas de S. Se $E'_k \geq S_{k+2}$ e $E'_k \geq E_k$ e $E'_{k+1} \geq E_{k+1}$, então, a solução S', obtida após trocar as tarefas j_k e j_{k+1}, será dominada pela solução S. Ou seja, S' não será melhor que S.*

Prova:

Sabe-se que após a realização da troca de j_k e j_{k+1}:

- o tempo de conclusão das tarefas anteriores a j_{k+1}, em S'_J, serão os mesmos que eram antes da troca
- quando $E'_k \geq S_{k+2}$ as tarefas subsequentes a j_k, em S'_J, não terão os tempos de conclusão melhorados

Assim, a única alternativa para melhorar a função objetivo é com relação ao tempo de conclusão das tarefas j_k e j_{k+1}. Sabe-se que antes da troca $F_k \geq E_k$ e que $F_{k+1} \geq E_{k+1}$. Sabe-se também que após a troca $F'_k \geq E'_k$ e $F'_{k+1} \geq E'_{k+1}$. Se $E'_k \geq E_k$ e $E'_{k+1} \geq E_{k+1}$ então $F'_k \geq E_k$ e $F'_{k+1} \geq E_{k+1}$. Ou seja, não haverá melhoria no tempo de fluxo dos lotes b_k e b_{k+1} e o fluxo ponderado total como um todo não irá melhorar $\square$.

Proposição 5 *Numa solução S, sejam j_k e j_{k+1}, $\forall k = 1,2,\ldots,N-1$, duas tarefas* ***consecutivas*** *na sequência de tarefas de S. Se $E'_k \geq S_{k+2}$ e $E_k \neq F_k$ e $E_{k+1} \neq F_{k+1}$, então, a solução S', obtida após trocar as tarefas j_k e j_{k+1}, será dominada pela solução S. Ou seja, S' não será melhor que S.*

Prova:

Sabe-se que:

- Se $E_k \neq F_k$ ($E_k < F_k$) então existe pelo menos uma tarefa do lote b_k processada após j_k. Em outras palavras, j_k não é a última tarefa processada do lote b_k e após a troca continuará não sendo a última.

- Se $E_{k+1} \neq F_{k+1}$ ($E_{k+1} < F_{k+1}$) então existe pelo menos uma tarefa do lote b_{k+1} processada após j_{k+1}. Em outras palavras, j_{k+1} não é a última tarefa processada do lote b_{k+1} e após a troca continuará não sendo a última.

Após a troca o tempo de conclusão das tarefas anteriores a j_{k+1}, em S'_J, serão os mesmos que eram antes da troca. E quando $E'_k \geq S_{k+2}$ as tarefas subsequentes a j_k, em S'_J, não terão os tempos de conclusão melhorados, incluindo o tempo de fluxo do lotes b_k e b_{k+1} já que j_k e j_{k+1} não são as últimas tarefas a serem processadas dos lotes b_k e b_{k+1}. Logo, o fluxo ponderado total como um todo não irá melhorar □.

Proposição 6 *Numa solução S, sejam j_k e j_{k+1}, $\forall k = 1,2,\ldots,N-1$, duas tarefas* ***consecutivas*** *na sequência de tarefas de S. Se $E'_k \geq S_{k+2}$ e $E_k = F_k$ e $E_{k+1} = F_{k+1}$ e $W_k(F'_k - F_k) + W_{k+1}(F'_{k+1} - F_{k+1}) \geq 0$, então, a solução S', obtida após trocar as tarefas j_k e j_{k+1}, será dominada pela solução S. Ou seja, S' não será melhor que S.*

Prova:

Sabe-se que:

- Quando $E_k = F_k$ significa que j_k é a última tarefa processada do lote b_k, antes e após a realização da troca

- Quando $E_{k+1} = F_{k+1}$ significa que j_{k+1} é a última tarefa processada do lote b_{k+1}, antes e após a realização da troca

Além disso, após a troca de j_k e j_{k+1}, se $E'_k \geq S_{k+2}$ então o tempo de conclusão das tarefas posteriores à tarefa j_k, em S'_J, não irão melhorar.

Quando se analisa o tempo de conclusão das tarefas posteriores à tarefa j_k, em S'_J, o melhor caso ocorre quando $E'_k = S_{k+2}$ ($E'_k \geq S_{k+2}$). Nesta situação, o tempo de conclusão das tarefas posteriores à tarefa j_k (em S'_J) permanecem inalterados e consequentemente o fluxo associado aos lotes correspondentes também não se alteram. Em S'_J, o tempo de conclusão das tarefas anteriores à tarefa j_{k+1} também não se alteram, de modo que o fluxo total da solução seja influenciado apenas pelos tempos de conclusão das tarefas j_k e j_{k+1}. Antes da troca, as parcelas associadas

aos lotes b_k e b_{k+1}, com respeito ao fluxo ponderado total, eram $W_k F_k$ e $W_{k+1} F_{k+1}$, respectivamente. Após a troca de j_k e j_{k+1} as parcelas serão $W_k F'_k$ e $W_{k+1} F'_{k+1}$, respectivamente. Logo, o valor da função objetivo será alterado pelo menos por $\Delta = W_k(F'_k - F_k) + W_{k+1}(F'_{k+1} - F_{k+1})$, que ocorre quando $E'_k = S_{k+2}$. Como o valor da função objetivo será influenciado apenas pelos tempos de conclusão das tarefas j_k e j_{k+1} se $\Delta \geq 0$ o valor da função objetivo não irá melhorar após a troca □.

3.4.4 Algoritmo *Iterated Local Search* (ILS) proposto

ILS (Lourenço et al., 2003) é uma metaheurística simples e de aplicação geral que usa busca local em modificações da solução corrente. Quatro métodos básicos são necessários para derivar um algoritmo ILS:

- CONSTRUCTION: é um método que retorna uma solução inicial viável S. Como dito anteriormente, a solução inicial foi gerada pelo procedimento descrito na Seção 3.4.2 (Algoritmo 5);
- LOCAL_SEARCH: é um procedimento de busca local que pode melhorar S_1 obtendo um ótimo local S_2. A busca local utilizada é baseada em movimentos de troca e foi descrita na Seção 3.4.3 (Algoritmo 6);
- PERTURBATION: é um método que perturba a solução corrente S levando a alguma solução intermediária S_1;
- ACCEPTANCE_CRITERION: é o critério de aceitação que decide para qual solução a próxima perturbação é aplicada.

Os métodos CONSTRUCTION e LOCAL_SEARCH do algoritmo ILS foram apresentados nas seções 3.4.2 e 3.4.3, respectivamente.

Optou-se por perturbar apenas a alocação de tarefas a lotes, não alterando o sequenciamento de tarefas. Remover a tarefa de um lote e colocá-la em outro já altera bastante o cálculo da função objetivo, pois pode influenciar tanto na parcela associada aos custos de entrega quanto no cálculo do fluxo ponderado total, já que para o cálculo do tempo de fluxo ponderado é necessário conhecer o tempo de término da última tarefa do respectivo lote. O pseudocódigo do procedimento de perturbação utilizado está descrito no Algoritmo 9 e será denominado apenas por PERTURBATION. Ele recebe dois parâmetros como entrada: uma solução S que será perturbada e d que é o número de perturbações a serem realizadas na solução

S. Para cada perturbação realizada na solução S, é sorteada uma posição k (linha 3) do sequenciamento de tarefas S_J de S. Seja j_k a tarefa sorteada da posição k, e que pertence a um cliente i e a um lote b qualquer. Esta tarefa j_k será alocada a um novo lote $newBatch$, $newBatch \neq b$, do mesmo cliente, sorteando-se um valor no intervalo $1..n_i$ (linhas 6 e 7).

Algoritmo 9: PERTURBATION (S, d)

```
   output: Perturbed solution S
1  begin
2      for cont := 1 to d do
3          k := RAND (1 ... N);
4          j_k := S.S_J[k]; // Get the job at position k
5          i := customer of job j_k ; // Get the customer i of job j_k
6          newBatch := RAND (1..n_i);
7          S.S_B[k] := newBatch;
8      return S
```

No algoritmo ILS proposto (veja o Algoritmo 10), o critério de aceitação está descrito entre as linhas 10-13. Se a solução S_2 (sobre a qual fora aplicado o procedimento LOCAL_SEARCH) é melhor que a solução corrente S, então S_2 substitui S (linhas 10 e 11) e se torna a nova solução corrente. Caso contrário, S_2 pode ser aceita com uma pequena probabilidade β, mesmo que seja pior que S (linhas 12 e 13).

Uma descrição geral do pseudocódigo do algoritmo proposto, chamado simplesmente de ILS, é apresentado no Algoritmo 10. O algoritmo tem três parâmetros de entrada: o parâmetro d que controla o nível da perturbação; o parâmetro β, que controla a probabilidade de aceitação da solução melhorada mesmo que seja pior que a solução corrente; e o parâmetro *numberSwap*, que é o número de trocas aleatórias avaliadas na solução no procedimento RPI_LS (veja o Algoritmo 8).

O procedimento CONSTRUCTION constrói uma solução inicial (linha 2) que é melhorada pelo procedimento de busca local (linha 3). As iterações do algoritmo ILS são computadas nas linhas 5-13, até que o critério de parada seja satisfeito. O critério de parada é baseado em um tempo máximo de execução, que foi fixado inicialmente em $500 \times N$ milissegundos, onde N é o número de tarefas da instância que está sendo avaliada. Durante cada iteração, a solução corrente S é perturbada (linha 6) pelo procedimento PERTURBATION, e melhorada pela busca local, obtendo uma solução S_2 (linha 7). Nas linhas 8 e 9, se a solução S_2 é melhor que a melhor solução obtida até o momento então é atualizada a melhor solução. Nas linhas 10-13 é testado o critério de aceitação. Se S_2 é melhor que a solução corrente S então

ela é aceita (S_2 substitui S), caso contrário ela pode ser aceita com uma pequena probabilidade β, de modo que haja um balanço entre diversificação e intensificação. A melhor solução encontrada durante todas as iterações é retornada pelo algoritmo (linha 14).

Algoritmo 10: ILS (d, β, *numberSwap*)

```
   output: Best founded solution S*
 1 begin
 2 |  S := CONSTRUCTION () ; // WSPT based
 3 |  S := LOCAL_SEARCH (S, numberSwap);
 4 |  S* := S; // Best solution
 5 |  while not stopping criterion do
 6 |  |  S1 := PERTURBATION (S, d);
 7 |  |  S2 := LOCAL_SEARCH (S1, numberSwap);
 8 |  |  if f(S2) < f(S*) then        /* Update the best solution */
 9 |  |  |  S* := S2;
10 |  |  if f(S2) ≤ f(S) then         /* The acceptance criterion */
11 |  |  |  S := S2;
12 |  |  else if RAND (0..1) ≤ β then
13 |  |  |  S := S2 ; // Accept worst solution
14 |  return S*
```

3.4.5 Algoritmo *Iterated Greedy* (IG) proposto

Iterated Greedy (IG) é uma simples e efetiva metaheuristica que foi introduzida recentemente por Ruiz & Stützle (2007) para resolver um problema de sequenciamento de tarefas em um ambiente *flowshop*, sendo inclusive estado da arte em diversos outros problemas de sequenciamento.

Um algoritmo IG começa com uma **solução inicial** e iterativamente são aplicados em sequência um procedimento de destruição e reconstrução. A fase de **destruição** remove aleatoriamente alguns componentes da solução incumbente S, levando a uma solução parcial S_p. Em seguida, uma nova solução completa S' é reconstruída de forma gulosa a partir de S_p. Essa é a chamada fase de **reconstrução**. Um **critério de aceitação**, ao final de cada iteração, define se a nova solução S' irá substituir a atual solução incumbente S. No final do processo é retornada a melhor solução encontrada. Opcionalmente pode ser adicionado um procedimento de busca local para melhorar a solução reconstruída S' antes de se avaliar o critério de aceitação (Pan & Ruiz, 2012).

A seguir estão descritos os principais componentes implementados desta metaheurística, que foi adaptada para se resolver o problema abordado neste trabalho.

As principais ideias utilizadas na implementação tiveram como motivação o trabalho realizado por Ruiz & Stützle (2008). Vale lembrar que a busca local e a heurística construtiva utilizadas na heurística IG proposta são exatamente as mesmas que foram utilizadas no ILS.

3.4.5.1 Destruição

O procedimento DESTRUCTION é responsável por destruir parcialmente uma solução. Este método possui 3 parâmetros de entrada: uma solução completa S, que é a solução que será parcialmente destruída; um parâmetro d, que é o número de elementos que serão removidos da solução S; e um conjunto π_R que irá guardar os elementos que foram removidos de S. O método DESTRUCTION retorna uma solução S alterada após a remoção aleatória de alguns de seus elementos. O conjunto π_R armazena os componentes removidos, na ordem de sua remoção, de tal modo que $\pi_R[1]$ é o primeiro elemento removido e $\pi_R[d]$ é o último elemento removido.

O pseudocódigo do procedimento DESTRUCTION pode ser visto no Algoritmo 11. Inicialmente é sorteada uma posição k do sequenciamento de tarefas S_J da solução S (linha 3). A seguir a tarefa j_k é armazenada no conjunto π_R (linhas 4 e 5) e é efetivamente removida de S_J (linha 6). Da mesma forma, deve-se remover da solução S o lote ao qual a tarefa j_k estava alocada (linha 7). Assim, tanto a permutação de tarefas S_J quanto o arranjo de lotes S_B terão o elemento da posição k efetivamente removidos da solução S. Este processo é repetido d vezes (linha 2). Ao final é retornada a solução parcial S com seus d componentes removidos.

Algoritmo 11: DESTRUCTION (S, d, π_R)

```
   output: Altered Solution S
1  begin
2      for cont := 1 to d do
3          k := RAND (1 ... |S.S_J|);
4          j_k := S_J[k]; // Get the job at position k
5          π_R.PUSH_BACK(j_k); // Add j_k to permutation of jobs
           removed π_R
6          S_J[k].REMOVE(); // Remove job of position k
7          S_B[k].REMOVE(); // Remove the respective batch at
           position k
8      return S
```

3.4.5.2 Reconstrução

Na fase de reconstrução da solução parcial foi utilizada uma heurística gulosa que é baseada na heurística construtiva NEH (Nawaz et al., 1983). A solução parcial é definida por $S_p = (S_p.S_J, S_p.S_B)$, onde $S_p.S_J$ e $S_p.S_B$ são, respectivamente, a permutação de tarefas (S_J) e arranjo de lotes (S_B) da solução S_p. No procedimento NEH, cada tarefa j em π_R é reinserida em todas as $|S_p.S_J| + 1$ possíveis posições de $S_p.S_J$, gerando $|S_p.S_J| + 1$ sequências parciais que incluem a tarefa j. Para cada sequência parcial gerada, a tarefa j é colocada no lote que gere o menor valor da função objetivo da sequência parcial. A sequência parcial com o melhor valor da função objetivo é mantida para a iteração seguinte. O processo termina quando π_R está vazio e S_p é uma solução completa.

O método de reconstrução, denominado de RECONSTRUCTION está descrito no Algoritmo 12. O método possui 2 parâmetros de entrada: uma solução parcial S_p e uma permutação π_R que contém todas as tarefas que foram removidas pelo procedimento DESTRUCTION. Inicialmente, é criada a solução S' que será retornada por este procedimento (linha 2). O processo iterativo continua até que não haja mais tarefas a serem inseridas na solução parcial, ou seja, até que $|\pi_R| = 0$ (linha 3). A primeira tarefa j de π_R é obtida e removida de π_R (linha 5). Esta tarefa é inserida em cada possível posição k da permutação de tarefas S_J da solução parcial S_p (linhas 7-8). Ao inserir a tarefa j na posição k é determinada a melhor alocação de lote para esta tarefa (linhas 9-14). Observe que na linha 15 é removida a tarefa j e o respectivo lote que haviam sido inseridos nas linhas 8-9. Nas linhas 12-14 é armazenada a melhor sequência parcial e alocação de lotes para cada inserção da tarefa j. A atribuição realizada na linha 16 é para que na próxima iteração seja utilizada a melhor sequência parcial gerada após inserir j. No final do procedimento é retornada uma solução completa S' (linha 17).

3.4.5.3 Estrutura do algoritmo *Iterated Greedy*

Uma descrição geral do pseudocódigo da heurística proposta baseada em *Iterated Greedy* é apresentada no Algoritmo 13. O algoritmo IG tem três parâmetros de entrada: o parâmetro d, que é o número de elementos que serão removidos da solução completa para gerar a solução parcial na etapa de destruição; o parâmetro β, que controla a probabilidade de aceitação da solução melhorada mesmo que seja pior que a solução corrente; e o parâmetro *numberSwap*, que é o número de trocas aleatórias avaliadas na solução no procedimento RPI_LS (veja o Algoritmo 8).

O procedimento CONSTRUCTION constrói uma solução inicial (linha 2) que é

Algoritmo 12: RECONSTRUCTION (S_p, π_R)

```
   output: Complete solution S'
 1 begin
 2     S' := ∅;
 3     while |π_R| > 0 do
 4         f* := +∞;
 5         j := π_R.FRONT(); // Get the first job of the π_R and
           remove it from π_R
 6         i := customer of job j ; // Get the customer i of job j
 7         foreach position k of S_p.S_J do
               /* Also consider the position after the last element
                  */
 8             Insert job j at position k of S_p.S_J;
 9             Insert a batch at position k of S_p.S_B; // Empty batch
10             for b := 1 to n_i do
11                 S_p.S_B[k] := b; // Put job j at batch b
12                 if f(S_p) < f* then
13                     f* := f(S_p);
14                     S' := S_p; // Remember the best position of
                       insertion and batch allocation for job j
15             Remove the elements inserted in positon k of S_p.S_J and S_p.S_B;
16         S_p := S';
17     return S'
```

melhorada pelo procedimento de busca local (linha 3). As iterações do algoritmo IG são computadas nas linhas 5-14 até que o critério de parada seja satisfeito. O critério de parada é baseado em um tempo máximo de execução, que foi fixado inicialmente em $500 \times N$ milissegundos, onde N é o número de tarefas da instância que está sendo avaliada. Durante cada iteração, a solução corrente S é destruída parcialmente (linha 6), pelo procedimento DESTRUCTION, gerando uma solução parcial S_p. Uma nova solução completa S_1, possivelmente diferente de S, é reconstruída pelo procedimento RECONSTRUCTION e melhorada pela busca local, obtendo uma solução S_2 (linhas 7-8). Nas linhas 9 e 10, se a solução S_2 é melhor que a melhor solução obtida até o momento então é atualizada a melhor solução. Nas linhas 11-14 é testado o critério de aceitação. Se S_2 é melhor que a solução corrente S então ela é aceita (S_2 substitui S), caso contrário ela pode ser aceita com uma pequena probabilidade β, de modo que haja um balanço entre diversificação e intensificação. A melhor solução encontrada durante todas as iterações é retornada pelo algoritmo (linha 15).

Algoritmo 13: IG (d, β, *numberSwap*)

```
   output: Best founded solution S*
 1 begin
 2 |  S := CONSTRUCTION ();
 3 |  S := LOCAL_SEARCH (S, numberSwap);
 4 |  S* := S; // Best solution
 5 |  while not stopping criterion do
 6 |  |  S_p := DESTRUCTION (S, d, π_R);
 7 |  |  S_1 := RECONSTRUCTION (S_p, π_R);
 8 |  |  S_2 := LOCAL_SEARCH (S_1, numberSwap);
 9 |  |  if f(S_2) < f(S*) then          /* Update the best solution */
10 |  |  |  S* := S_2;
11 |  |  if f(S_2) ≤ f(S) then           /* The acceptance criterion */
12 |  |  |  S := S_2;
13 |  |  else if RAND (0..1) ≤ β then
14 |  |  |  S := S_2;
15 |  return S*
```

3.5 Análise de desempenho computacional e estatística

Os algoritmos ILS e IG foram codificados em C++ e executados em uma máquina com processador Intel(R) Xeon(R) CPU X5650 @ 2.67GHz e 48 GB de RAM. A exploração do paralelismo foi utilizado somente na execução do modelo matemático no *solver* CPLEX versão 12.4.

Para obter a solução ótima para algumas instâncias, foram reproduzidos os experimentos realizados por Mazdeha et al. (2012). Estes autores propuseram um algoritmo *Branch and Bound* (B&B). O algoritmo B&B foi reimplementado segundo as diretrizes do artigo original, já que não foi possível obter a implementação diretamente por solicitação ao autor do trabalho supracitado. No entanto, algumas propriedades e pontos importantes da implementação não estão perfeitamente claros no artigo original, de modo que não se conseguiu provar a *otimalidade* das soluções obtidas por este algoritmo. Então, as soluções fornecidas pelo B&B são utilizadas como limitantes superior nas comparações. Possíveis pontos de divergência da implementação original estão descritos no Apêndice D, para que possam, se necessário, serem reproduzidos por outros autores. O código fonte bem como as instância utilizadas também podem ser obtidas via solicitação por correio eletrônico ao autor do presente trabalho.

Foi utilizada a metodologia de Desenho de Experimentos (DOE)

(Montgomery, 2006) e a ferramenta de Análise de Variância (ANOVA) na condução dos experimentos e análise de resultados. Foram verificadas as três principais hipóteses da ANOVA paramétrica: normalidade, homocedasticidade e independência residuais. Por questão de clareza na exposição dos resultados, a verificação das pressuposições da ANOVA estão organizadas no Apêndice C deste trabalho.

Nas seguintes subseções estão descritos os principais experimentos realizados. As seções estão subdivididas por experimento, sendo que cada um deles foi realizado com uma hipótese específica.

3.5.1 Métrica para avaliação dos algoritmos

Tanto na calibração de parâmetros das heurísticas quanto nos experimentos de comparação foi utilizada a medida do Desvio Percentual Relativo (RPD-do inglês *Relative Percentage Deviation*). Para uma instância i, o RPD é calculado da seguinte forma (Vallada et al., 2008):

$$RPD(i)\% = \frac{f_{method}(i) - f_{best}(i)}{f_{best}(i)} \times 100\% \tag{3.30}$$

onde $f_{method}(i)$ é a solução (valor da função objetivo) obtido por um dado método (algoritmo ou configuração de parâmetro) e $f_{best}(i)$ é a melhor solução obtida entre todos os métodos comparados.

Nem sempre comparar com a solução ótima é viável uma vez que o valor ótimo pode não ser encontrado para todas as instâncias utilizadas. Por isso, serão realizadas comparações com a melhor solução conhecida (possivelmente não-ótima). Mas, depois também será feita uma análise considerando algumas soluções provadas serem ótimas através do modelo matemático do problema.

3.5.2 Geração de problemas teste

As instâncias do problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$ foram geradas aleatoriamente de acordo com o trabalho de Mazdeha et al. (2012), uma vez que não há um *benchmark* de problemas testes disponível. Foram considerados problemas testes com o número de tarefas N variando de 5 a 40 aleatoriamente distribuídas entre os clientes. O número de clientes F varia de 2 a 10. O tempo de processamento (p_{ij}), o peso (w_{ij}) e a data de liberação (r_{ij}) das tarefas variam no intervalo de número inteiros $[1, 100]$, [1,10] e [1,100], respectivamente. A dificuldade para se resolver o problema pode ser influenciada pela interação entre os custos de entrega e o fluxo ponderado das tarefas. Então, na geração dos problemas foram consideradas duas classes de

problemas: na **classe A**, o custo de entrega de cada cliente é um inteiro escolhido aleatoriamente no intervalo $[1001, 10000]$, que é um valor alto se comparado ao tempo de processamento; na **classe B**, o custo de entrega considera um valor no intervalo $[1, 1000]$. Além disso, para tornar os resultados representativos, foram geradas 10 instâncias para cada configuração de problema teste.

As distribuições do número de tarefas N e do número de cliente F utilizados na geração dos problemas da classe A estão especificadas na Tabela 3.2 enquanto que os da classe B são mostrados na Tabela 3.3. Como pode ser visto nestas tabelas foram geradas 780 problemas testes, sendo 320 da classe A e 460 da classe B. Vale ressaltar que como mencionado no trabalho Mazdeha et al. (2012) as instâncias devem obedecer a seguinte restrição: para cada par de tarefas j e j' sempre que $\frac{p_j}{w_j} \leq \frac{p_{j'}}{w_{j'}}$ então $r_j \leq r_{j'}$ e também cada cliente deve estar associado a pelo menos 2 tarefas.

As instâncias geradas de acordo com (Mazdeha et al., 2012), que são as 780 instâncias mencionadas anteriormente, serão denominadas neste trabalho de instâncias de pequeno e médio porte porque em outros experimentos, que serão apresentados adiante, foram consideradas outras distribuições de N e F que consideram valor de N com até 100 tarefas e valores de F com até 10 clientes. Estas últimas instâncias serão chamadas de instâncias de grande porte

Tabela 3.2. Configurações utilizadas na geração de problemas testes da classe A, de acordo com o trabalho de Mazdeha et al. (2012)

N	**F**	**# Instâncias**	**N**	**F**	**# Instâncias**
5	2	10	17	2	10
7	2	10	17	3	10
7	3	10	17	4	10
8	4	10	17	5	10
10	2	10	17	7	10
10	3	10	20	3	10
10	4	10	20	4	10
10	5	10	20	5	10
12	4	10	20	7	10
13	2	10	20	10	10
13	3	10	23	4	10
14	7	10	23	5	10
15	2	10	23	7	10
15	3	10	23	10	10
15	4	10	25	7	10
15	5	10	25	10	10
Total		160	**Total**		160

Tabela 3.3. Configurações utilizadas na geração de problemas testes da classe B, de acordo com o trabalho de Mazdeha et al. (2012)

N	F	# Instâncias	N	F	# Instâncias
5	2	10	20	4	10
7	2	10	20	5	10
7	3	10	20	7	10
8	4	10	20	10	10
10	2	10	22	2	10
10	3	10	22	3	10
10	4	10	23	4	10
10	5	10	23	5	10
12	4	10	23	7	10
13	2	10	23	10	10
13	3	10	25	4	10
14	7	10	25	5	10
15	2	10	25	7	10
15	3	10	25	10	10
15	4	10	27	5	10
15	5	10	27	7	10
17	2	10	27	10	10
17	3	10	30	7	10
17	4	10	30	10	10
17	5	10	35	7	10
17	7	10	35	10	10
20	2	10	40	7	10
20	3	10	40	10	10
Total		230	**Total**		230

3.5.3 Calibração de parâmetros do ILS algoritmo proposto para instâncias de pequeno e médio porte

Nos experimentos de calibração do algoritmo ILS foi utilizado um conjunto independente de instâncias, gerado da mesma forma que descrito na Seção 3.5.2, num total de 780 problemas testes. A razão é simples: utilizar um mesmo *benchmark* para a calibração de parâmetros e comparação dos métodos poderia levar a resultados tendenciosos.

Vale lembrar que como explicado no Algoritmo 10, o algoritmo ILS proposto possui três parâmetros de entrada: d, β e *numberSwap*. Os valores testados para cada um dos parâmetros estão especificados na Tabela 3.4. Observe que há um total de 54 combinações de parâmetros do ILS a serem avaliadas. Cada um dos 780 problemas testes foi resolvido cinco vezes para cada uma das 54 combinações

de parâmetros, de modo que para este experimento tem-se um total de 210600 observações. O critério de parada utilizado é baseado em um tempo máximo de CPU igual a $500 \times N$ milissegundos. Os resultados foram analisados através do RPD% (calculado conforme equação (3.30)) e ANOVA paramétrica.

Tabela 3.4. Conjunto de valores testados para a calibração do ILS, considerando somente as instâncias de pequeno e médio porte

Parâmetro	**Valores testados**	**# Valores**
d	$\{1; 2; 3; 4; 5; 6\}$	6
β	$\{0{,}0; 0{,}5; 1{,}0\}$	3
$numberSwap$	$\{N; 2N; N^2\}$	3
Total		54

A variável de resposta considerada na ANOVA é o RDP% médio, enquanto que os tratamentos são as configurações dos parâmetros consideradas na calibração. Foram verificadas as três pressuposições da ANOVA, a saber igualdade de variância, normalidade e independência dos resíduos (ver Apêndice C.1). O resultado da ANOVA para este experimento pode ser visto na Tabela 3.5.

Nesta tabela, o *teste-F* na tabela ANOVA verifica se há qualquer diferença significante entre as médias do RPD% encontradas por cada uma dos tratamentos utilizadas na calibração do ILS. Esta tabela decompõe variação total entre todas as observações, na variação devido ao efeito dos tratamentos e na variação devida ao acaso ou resíduos (veja coluna "Fonte da Variação"). O número de graus de liberdade (veja coluna "'Graus de Liberdade") dos tratamentos é igual a $I-1$, onde I é o número de tratamentos utilizados. No caso, $I = 54$ configurações, sendo o número de graus de liberdade associado aos tratamentos igual a 53. Já o número de graus de liberdade associados aos resíduos é igual a $I(J-1)$, onde J é o número de repetições. No caso, $J = 5$ repetições o que dá um total de 216 graus de liberdade. A coluna "Quadrados Médios" corresponde ao quociente entre a soma de quadrados pelo grau de liberdade da respectiva fonte de variação. O *valor de F*, que neste caso é igual a 4476,67, é o quociente entre o quadrado médio dos tratamentos pelo quadrado médio do resíduos. Uma vez que o *valor-P* do teste-F, que é o valor de interesse, é menor que 0,05, há uma diferença estatisticamente significante de uma das configurações para outra, com um nível de confiança de 95%.

A ANOVA mostrou que há diferença significativa entre as configurações, mas não especifica qual configuração é a melhor ou não especifica onde estão os contrastes entre os tratamentos. Para este fim, foi conduzido um teste de Comparações Múltiplas. Na Figura 3.6 está o gráfico de médias resultante do teste *Tukey* da

Tabela 3.5. Tabela ANOVA para comparação das configurações testadas na calibração do algoritmo ILS, considerando somente as instâncias de pequeno e médio porte

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	329,10	53	6,21	**4476,67**	**0,00**
Resíduos	0,30	216	0,00		
Total	329,40	269			

Diferença Honestamente Significativa HSD com nível de confiança de 95% para as configurações testadas do algoritmo ILS. Das 54 configurações avaliadas, por questão de facilidade de visualização, somente as configurações de 0 a 19 são exibidas neste gráfico. Por esta figura é possível ver que existe diferença significativa entre as configurações, pois há diversos intervalos que não se sobrepõe, mesmo embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. A configuração 19 apresentou a melhor média e corresponde aos valores $d = 3$; $\beta = 0{,}0$ e $numberSwap = 2N$. Observe que nesta configuração o parâmetro β possui valor 0,0, indicando que, para esta implementação, não se deve considerar soluções de qualidade inferior a solução corrente no critério de aceitação, pelo menos para este conjunto de instâncias.

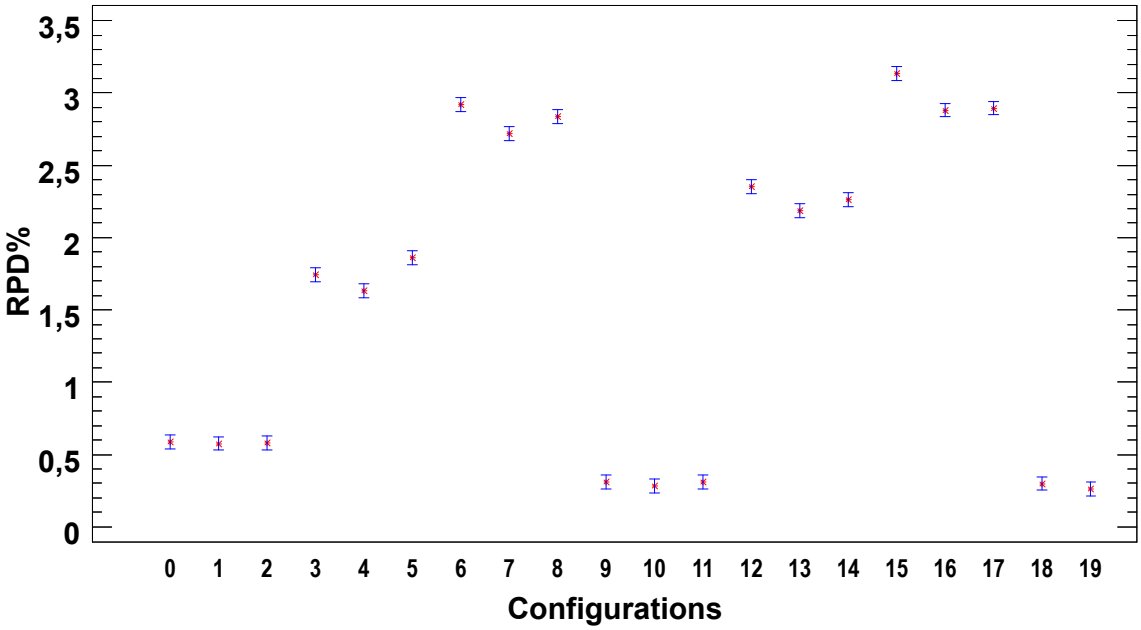


Figura 3.6. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 54 configurações de ILS avaliadas

3.5.4 Experimentos com o modelo de Programação Linear Inteira Mista

3.5.4.1 Experimento 1 - Utilização das restrições de remoção de simetria e da solução inicial

Neste experimento, pretende-se verificar se a utilização das restrições de remoção de simetria, especificadas pelo conjunto de restrições (3.28), (3.29) mostrados na Seção 3.3.3, realmente foram eficazes ao melhorar a convergência do método exato.

A outra hipótese a ser verificada é se a utilização de uma solução inicial viável no MILP realmente ajuda o método exato a encontrar soluções em um tempo computacional menor. Optou-se por utilizar como solução inicial a solução gerada pelo algoritmo ILS proposto. O algoritmo ILS foi executado durante $500 \times N$ milissegundos e com os parâmetros configurados de acordo com os valores obtidos na calibração: $d = 3$; $\beta = 0{,}0$ e $numberSwap = 2N$. A solução obtida pelo ILS, neste tempo, é, então, passada como um limitante superior ao ILOG/CPLEX, de tal modo que ele já comece com uma solução inicial viável.

Neste experimento, optou-se por utilizar todas as 20 instâncias com 8 tarefas que em experimento prévio realizado, já utilizando-se das restrições de remoção de simetria acima citadas, mostraram-se possíveis de se resolver de forma ótima em um tempo computacional razoável para execução com repetições. A utilização de repetições é necessário para comparação através de métodos estatísticos como ANOVA ou teste de *Kruskal Wallis*. Cada uma das 20 instâncias foi resolvida 30 vezes e o tempo gasto para provar a solução ótima é registrado e será utilizado como variável de resposta nos experimentos estatísticos. Em cada uma das 30 repetições o algoritmo ILS é executado uma única vez e a solução obtida é fornecida como limitante superior para o método exato.

Considera-se, então, três versões do modelo de acordo com a utilização das restrições de remoção de simetria e da utilização da solução inicial:

- Utilização de simetria e utilização da solução inicial. Esta versão simplesmente será mencionada como "$+simet + initSol$".
- Utilização de simetria e não utilização da solução inicial. Esta versão simplesmente será mencionada como "$+simet - initSol$".
- Não utilização de simetria e não utilização da solução inicial. Esta versão simplesmente será mencionada como "$-simet - initSol$".

Cada uma das três versões resolve as 20 instâncias 30 vezes e o tempo médio, considerando as 600 observações, que cada versão gastou para alcançar a solução ótima estão especificados na Tabela 3.6. Observa-se que a versão $+simet + initSol$ obteve na média os melhores valores de tempo e é um indicativo de que a inclusão das restrições de remoção de simetria e a utilização da solução inicial podem ajudar o modelo a convergir para o ótimo em um menor tempo. É importante notar que mesmo que a solução fornecida pelo algoritmo ILS ao *solver* seja a solução ótima, a prova de otimalidade por parte do modelo não é trivial.

Tabela 3.6. Tempo médio gasto para cada versão do modelo encontrar a solução ótima

Versão	**Tempo Médio (s)**
$+simet + initSol$	**19,15**
$+simet - initSol$	21,61
$-simet - initSol$	107,11

Para validar estatisticamente os resultados obtidos pelas três versões e verificar se as diferenças observadas são estatisticamente significantes, foi realizada uma Análise de Variância (ANOVA) paramétrica. As hipóteses para o teste são as seguintes:

- H_0: $m_{\text{+simet +initSol}} = m_{\text{+simet -initSol}} = m_{\text{-simet -initSol}}$, ou, em outras palavras, todos os possíveis contrastes entre as médias dos tempos das versões do modelo são estatisticamente nulos, no nível de significância considerado. Neste caso, $m_{version}$ é o tempo médio gastro pela versão $version$ considerada do modelo
- H_1: não H_0, ou, em outras palavras, existe pelo menos um contraste entre as médias dos tempos das versões do modelo diferente de zero, no nível de significância considerado

Ao se testar os pressupostos da ANOVA paramétrica não foi verificada a igualdade de variância. Por isso, optou-se por utilizar o teste de *Kruskal Wallis* que é uma alternativa não-paramétrica para o caso onde a ANOVA paramétrica não pode ser aplicada. As hipóteses do teste são exatamente as mesma mencionadas anteriormente. Os resultados deste teste estão especificados na Tabela 3.7. Uma vez que o *valor-P* do teste, que é o valor de interesse, é menor que 0,05, há uma diferença estatisticamente significante de uma das versões para outra, com um nível de confiança de 95%.

O teste de *Kruskal Wallis* não especifica quais versões são diferentes entre si, de tal modo que foi realizado um teste de Comparações Múltiplas para comparar cada

Tabela 3.7. Teste de *Kruskal Wallis* para comparação das médias de tempo de três versões consideradas do modelo.

Versão	**Tamanho da Amostra**	***Average Rank***
$+simet + initSol$	30	15,97
$+simet - initSol$	30	45,03
$-simet - initSol$	30	75,50
Estatística do teste = 77,91; **valor-P = 0**		

par de médias com um nível de confiança de 95%. A Tabela 3.8 mostra o resultado deste teste. 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 significa que as versões são significativamente diferentes no nível de confiança selecionado de 95%. Na tabela, a existência de diferença é indicado por um (*) na coluna "Significante". Pode-se ver que existe diferença estatística significativa entre todos os pares de médias considerados. A análise do primeiro par de versões ($+simet + initSol$, $+simet - initSol$) é um indicativo de que a utilização de uma solução inicial no modelo produz um melhoramento no tempo necessário para encontrar a solução ótima. E considerando-se o último par ($+simet - initSol$, $-simet - initSol$) conclui-se que a utilização das restrições de remoção de simetria acarretam um ganho médio em tempo muito satisfatórios. Ou seja, é viável a utilização da solução inicial e, principalmente, a utilização das restrições de remoção de simetria.

Tabela 3.8. Teste de comparações múltiplas para o tempo médio de execução da versões do modelo

Par de algoritmos	**Significante**	**Diferença**	**+/-Limite**
$+simet + initSol$, $+simet - initSol$	(*)	-2,46	0,98
$+simet + initSol$, $-simet - initSol$	(*)	-87,96	0,98
$+simet - initSol$, $-simet - initSol$	(*)	-85,50	0,98

A mesma análise pode ser vista na Figura 3.7. Esta figura mostra o gráfico de médias resultantes do teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95%. Uma vez que o intervalo para as versões não sobrepõe os outros intervalos, a média de tempo da versão $+simet+initSol$ é significativamente menor das médias das outras, mostrando que ela tem desempenho superior.

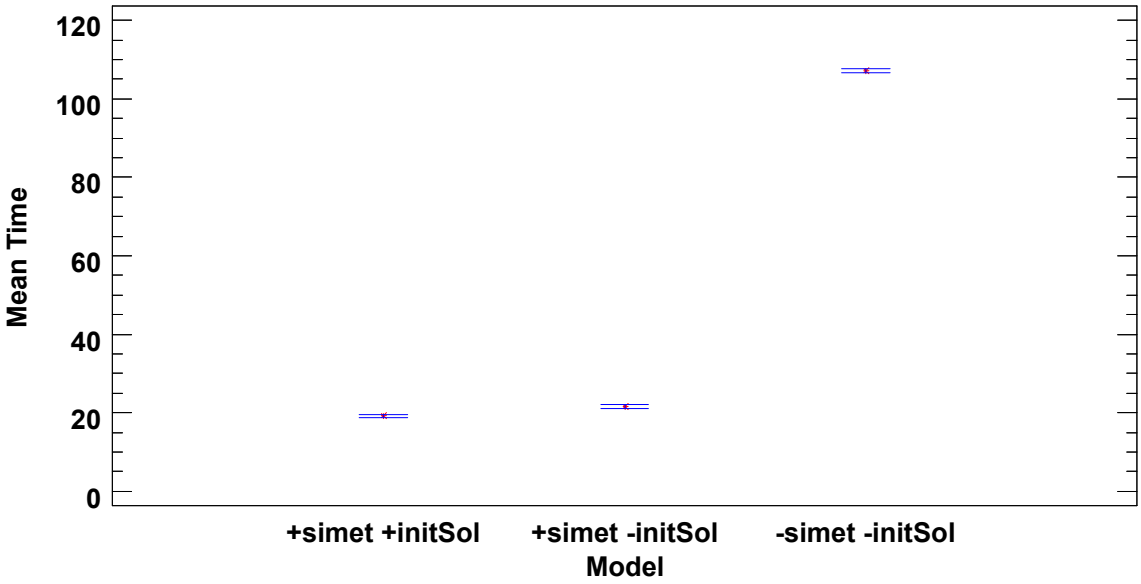


Figura 3.7. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos três versões do MILP.

3.5.4.2 Experimento 2 - Execução das instâncias teste no *solver* ILOG/CPLEX

Este experimento tem como intuito apenas a execução das instâncias no *solver* ILOG/CPLEX para tentar provar a solução ótima de algumas instâncias para posterior comparação com algoritmo ILS. O modelo foi executado uma única vez para cada problema teste. Limitou-se a 2 horas de tempo de execução (tempo de relógio) para se resolver cada problema teste. A máquina utilizada nos experimentos é a mesma utilizada nos experimentos anteriores, com exceção agora de estar utilizando todos os 12 núcleos (em paralelo) para se resolver o modelo para cada instância. O limite de 2 horas foi estabelecido para que fosse possível obter ou soluções ótimas ou um limitante superior para todas as 780 instâncias de modo que os resultados pudessem ser comparados com as heurísticas propostas e com o B&B de Mazdeha et al. (2012). A versão utilizada para a execução do modelo foi $+simet+initSol$ que no experimento mostrado na Seção 3.5.4.1 demostrou-se ser a melhor. Lembre que esta versão também utiliza a heurística ILS que roda durante $500 \times N$ milissegundos antes de fornecer a solução gerada para o modelo matemático.

Os resultados gerais quanto ao número de soluções ótimas com respeito a classe de problema A ou B são mostrados na Tabela 3.9. A entrada (*, *, A) corresponde a todas as instâncias da classe A e (*, *, B) corresponde a todas as instâncias da classe B. A partir dos dados mostrados nesta tabela, pode-se presumir que os problemas da classe B são mais difíceis de serem resolvidos já que o *solver* somente encontrou o

ótimo em 47/460 (10,22%) das instâncias desta classe, enquanto que para a classe A foram provados ótimos em 65/320 (20,31%) problemas testes. Através do modelo foi possível provar o ótimo para 112/780 (14,36%) instâncias. A coluna "*Gap* Médio" mostra o valor médio da diferença entre o limitante inferior e limitante superior obtidos pelo ILOG/CPLEX. Na média foi alcançado um *gap* de 45,85%. Observe, no entanto, que o *gap* para instâncias da classe B foi de 55,21%. Provavelmente, o *gap* da classe B foi maior que o da classe A porque na classe B as instância possuem tamanho maior de clientes e tarefas (reveja as Tabelas 3.2 e 3.3).

Tabela 3.9. Resultados gerais da execução do modelo matemático quanto à classe (A ou B) das instâncias.

Classe de problemas	**# Instâncias**	**# Ótimos provados**	***Gap* Médio**
(*, *, A)	320	65	32,40%
(*, *, B)	460	47	55,21%
Total	780	112	

A Tabela 3.10 mostra os resultados considerando agora somente o número de clientes F. A entrada (*, f, *) corresponde a todas as instâncias com o número de clientes $F = f$. É possível observar que a partir de 5 clientes o modelo não encontrou mais nenhum ótimo. Isso é um indicativo de que o número de clientes influencia na dificuldade de se encontrar soluções ótimas para o problema.

Tabela 3.10. Resultados da execução do modelo matemático considerando apenas o número de clientes.

Classe de problemas	**# Instâncias**	**# Ótimos provados**	***Gap* Médio**
(*, 2, *)	140	45	36,23%
(*, 3, *)	130	25	40,49%
(*, 4, *)	150	28	39,79%
(*, 5, *)	120	14	46,97%
(*, 7, *)	140	0	56,04%
(*, 10, *)	100	0	59,79%

Quanto ao número de tarefas N, a Tabela 3.11 mostra os resultados. Cada entrada (n, *, *) corresponde a todas as instâncias com o número de cliente $N = n$. Novamente o número de soluções ótimas quando se aumenta o número de tarefas decresce rapidamente, sendo que a partir de instâncias com 12 tarefas quase nenhum ótimo é encontrado. É importante notar que para instâncias com 5, 7 e 8 tarefas o modelo conseguiu provar o ótimo para todas elas. Para aquelas com 10 tarefas o ótimo foi provado para 38,75% delas.

Tabela 3.11. Resultados da execução do modelo matemático considerando apenas o número de tarefas.

Classe de problemas	# Instâncias	# Ótimos provados	*Gap* Médio
(5, *, *)	20	20	0,00%
(7, *, *)	40	40	0,00%
(8, *, *)	20	20	0,00%
(10, *, *)	80	31	16,70%
(12, *, *)	20	0	34,68%
(13, *, *)	40	0	46,69%
(14, *, *)	20	0	32,50%
(15, *, *)	80	1	46,74%
(17, *, *)	100	0	50,03%
(20, *, *)	110	0	54,33%
(22, *, *)	20	0	70,59%
(23, *, *)	80	0	57,68%
(25, *, *)	60	0	61,16%
(27, *, *)	30	0	71,14%
(30, *, *)	20	0	74,74%
(35, *, *)	20	0	77,28%
(40, *, *)	20	0	81,34%

Como dito anteriormente, foi passado um limitante superior ao modelo matemático. Esta solução foi gerada através de uma única execução da metaheuristica ILS. Em 467/780 (59,87%) problemas testes, o ILOG/CPLEX não conseguiu melhorar o limitante superior gerado por ILS, enquanto que em 313/780 (40,13%) instâncias houve melhoria. Foi calculado também o quanto a solução fornecida pelo ILS (f_{ILS}) difere daquela encontrada pelo método exato (f_{CPLEX}), calculando-se o desvio relativo conforme equação (3.31). Os resultados estão na Tabela 3.12 e indicam que a solução encontrada pelo ILS difere muito pouco da melhor solução encontrada pelo ILOG/CPLEX, em duas horas de execução, com desvio abaixo de 1%. O desvio médio considerando todas as instâncias foi de 0,34%.

$$\frac{f_{ILS} - f_{CPLEX}}{f_{CPLEX}} \times 100\% \tag{3.31}$$

3.5.5 Comparativo entre ILS, *Branch and Bound* e MILP

Como dito anteriormente, foi reimplementado o algoritmo *Branch and Bound* de modo que os experimentos realizados por Mazdeha et al. (2012) fossem repetidos e utilizados para comparação com o método exato e as heurísticas propostas neste trabalho.

Tabela 3.12. Desvio médio entre o limitante superior fornecido ao CPLEX em relação a melhor solução encontrada pelo CPLEX.

N	Desvio médio
5	0,00%
7	0,05%
8	0,02%
10	0,22%
12	0,14%
13	0,35%
14	0,47%
15	0,41%
17	0,64%
20	0,55%
22	0,05%
23	0,42%
25	0,38%
27	0,03%
30	0,03%
35	0,05%
40	0,12%

Para o B&B, cada umas das 780 instâncias foi executada por um tempo (de relógio) máximo de 24 horas, de tal forma que a melhor solução encontrada neste limite de tempo será utilizada para comparação com as soluções do modelo matemático e da heurística proposta. O MILP foi executado no ILOG/CPLEX durante um tempo máximo de 2 horas (tempo de relógio) utilizando-se 12 núcleos, totalizando o mesmo número de horas de execução que o B&B caso fosse executado com somente um núcleo. O ILS foi rodado durante um tempo máximo de $500 \times N$ milissegundos.

Com os dados experimentais obtidos pelas implementações ILS, B&B e MILP foi feita uma análise comparativa entre as melhores soluções encontradas por cada uma destas três implementações. Como dito anteriormente, comparar contra a solução ótima não é viável já que o valor ótimo pode não ser encontrado para todas as instâncias utilizadas. Por isso, serão realizadas comparações contra a melhor solução conhecida (possivelmente não-ótima).

Foi realizada a comparação do Desvio Percentual Relativo médio de cada uma das implementações. O desvio é calculado utilizado a melhor solução conhecida de cada implementação. Para o ILS a solução escolhida para comparação é a melhor dentre todas encontradas nas 30 repetições. O RPD% é calculado de acordo com a equação (3.30). As Tabelas 3.13 e 3.14 mostram, respectivamente, o resultado do RPD% médio para as classes de problemas A e B, considerando-se somente o número

de tarefas. Por exemplo, a entrada com número de tarefas igual a 5 corresponde a média do RPD% calculado para todas as instâncias com o número de tarefas igual a 5, para a classe de problemas em questão. É possível notar que para a classe de problemas A, aparentemente o B&B possui os melhores resultados com respeito a qualidade das soluções, com o RPD% médio sempre abaixo ou igual a 0,01%. No entanto, todas as 3 implementações se comportam de forma semelhante com RPD% médio muito baixo, sendo que o pior valor médio de RPD% foi de 1,02% que é atribuído ao MILP quando se considera instâncias com 23 tarefas.

Tabela 3.13. Desvio Percentual Relativo médio com relação à melhor solução conhecida. Problemas da classe A

N	MILP	ILS	B&B
5	**0,00**%	**0,00**%	**0,00**%
7	**0,00**%	0,08%	**0,00**%
8	**0,00**%	**0,00**%	**0,00**%
10	**0,00**%	0,14%	**0,00**%
12	**0,00**%	0,12%	**0,00**%
13	0,09%	0,08%	**0,01**%
14	0,01%	0,11%	**0,00**%
15	0,07%	0,05%	**0,00**%
17	0,32%	0,09%	**0,00**%
20	0,61%	0,07%	**0,00**%
23	1,02%	0,08%	**0,00**%
25	0,39%	0,08%	**0,01**%

Para as instâncias da classe B, que mostraram-se mais difíceis de serem resolvidas (conforme foi demonstrado na Seção 3.5.4.1), os menores valores de RPD% médio foram encontrados pela implementação ILS.

Vale ressaltar que a utilização de ferramentas estatísticas, como ANOVA, seria o ideal para se comparar estas 3 implementações e ter um resultado estatisticamente significante. Mas, de fato, isso é inviável de ser feito dado que o tempo de execução com repetições de todas as instâncias tanto do método exato quanto do B&B inviabilizariam o experimento, como pode ser visto na Tabela 3.15. Observe que, a partir de 12 tarefas, o limite de 2 horas de execução muitas vezes é excedido pelo MILP. A partir de 12 tarefas a memória disponível nem sempre era suficiente para execução do *solver* e, por isso, o tempo médio, algumas vezes, está abaixo de 7200 segundos. E para o B&B, a partir de 20 tarefas, o tempo para realizar o experimento com repetições se tornaria extremamente alto. O tempo gasto pela implementação ILS varia de acordo com o tamanho da instância, sendo que o menor valor é de 2,5

Tabela 3.14. Desvio Percentual Relativo (RPD) médio com relação à melhor solução conhecida. Classe de problemas B.

N	MILP	ILS	B&B
5	**0,00%**	**0,00%**	**0,00%**
7	**0,00%**	**0,00%**	**0,00%**
8	**0,00%**	0,04%	**0,00%**
10	**0,00%**	**0,00%**	**0,00%**
12	**0,00%**	**0,00%**	**0,00%**
13	0,05%	**0,00%**	0,06%
14	0,02%	**0,00%**	**0,00%**
15	0,05%	**0,00%**	**0,00%**
17	0,01%	**0,00%**	**0,00%**
20	0,06%	**0,00%**	0,04%
22	0,14%	**0,00%**	0,01%
23	0,04%	**0,00%**	**0,00%**
25	0,11%	**0,00%**	**0,00%**
27	0,10%	**0,00%**	0,01%
30	0,10%	**0,00%**	0,01%
35	0,30%	**0,00%**	0,18%
40	0,46%	**0,00%**	0,42%

segundos (instâncias com 5 tarefas) e o maior valor é de 20 segundos (instâncias com 40 tarefas).

É de suma importância a comparação do método heurístico ILS com o método exato, para ver a sua eficácia em encontrar soluções tão boas quanto a ótima. Serão mostrados a seguir alguns dados relativos à comparação das 112 soluções garantidamente ótimas (provadas pelo método exato) com as melhores soluções obtidas pelo algoritmo ILS e com a média das soluções obtidas por ILS. Para a comparação com o valor ótimo, não será mais mostrado o desvio e sim o erro (percentual) relativo já que se conhece o valor ótimo. O erro é calculado conforme a equação (3.32), onde f_{ILS} será, dependendo do contexto, ou a melhor solução encontrada pela heurística ILS ou a média das soluções encontradas por ILS, considerando-se em ambos os casos os resultados das 30 repetições, e f_{OPT} é a solução ótima obtida através do método exato. Os dados originais com os valores das soluções ótimas para cada instância estão anexados no Apêndice C.4.

$$\frac{f_{ILS} - f_{OPT}}{f_{OPT}} \times 100\% \tag{3.32}$$

Dos 112 ótimos provados pelo MILP, o ILS encontrou o mesmo valor ótimo, pelo menos 1 vez nas 30 repetições, em 99 destas instâncias. Somente em 13 ins-

Tabela 3.15. Tempo médio (segundos) gasto pelo MILP e B&B de acordo com o número de tarefas.

N	MILP	B&B
5	0,18	0,00
7	13,12	0,00
8	20,91	0,00
10	5057,95	0,01
12	6967,99	0,04
13	7200,18	0,55
14	7200,28	0,02
15	7173,54	2,45
17	7200,15	21,95
20	7224,02	415,74
22	6400,18	14473,50
23	7251,36	1168,58
25	7200,16	8856,69
27	7200,24	22232,36
30	6946,32	35768,48
35	6997,78	80959,75
40	6927,91	86400,01

tâncias o ILS não encontrou a solução ótima. O erro médio entre a melhor solução encontrada pelo ILS e a solução ótima foi de 0,06%, ao passo que o erro médio entre a médias das encontradas por ILS foi de 0,19%. Considerando-se apenas as instâncias em que o algoritmo ILS não conseguiu encontrar a solução ótima, o erro médio com respeito a melhor solução obtida pelo ILS foi de 0,54% e ao considerar a média das soluções tem-se um erro médio de 0,71%. Estes resultados mostram que o algoritmo proposto ILS é muito efetivo ao encontrar soluções de "boa" qualidade.

3.5.6 Experimentos realizados com instâncias de grande porte

Nesta seção serão apresentados os resultados dos experimentos realizados com instâncias de grande porte, que consideram problema com até 100 tarefas e 10 clientes. Primeiramente, na seção 3.5.6.1 está descrito como foram geradas estas instâncias. Na seção 3.5.6.2 estão os experimentos para verificar como as regras de dominância conseguiram melhorar a eficiência da busca local API_LS. Na seções 3.5.6.3 e 3.5.6.4 estão, respectivamente, os experimentos de calibração das heurísticas ILS e IG para as instâncias de grande porte. Uma comparação focando na qualidade das soluções obtidas pelos algoritmos ILS e IG é realizada na seção 3.5.6.5. Por fim, na seção 3.5.6.6 é avaliado o desempenho dos algoritmos ILS e IG quanto ao tempo de

execução.

3.5.6.1 Geração das instâncias de grande porte

Foi gerado outro conjunto de instâncias semelhante àquelas geradas na Seção 3.5.2. Os problemas gerados são de certa forma mais genéricos que os gerados por Mazdeha et al. (2012), porque não pressupõem que para cada par de tarefas j e j' sempre que $\frac{p_j}{w_j} \leq \frac{p_{j'}}{w_{j'}}$ então $r_j \leq r_{j'}$, como realizado por Mazdeha et al. (2012). O tempo de processamento, data de liberação, pesos e custos de entrega são gerados com as mesmas distribuições daquelas que foram usadas na Seção 3.5.2. O número de tarefas e clientes utilizados na geração das instâncias de grande porte são mostrados na Tabela 3.16. Foram geradas 10 instâncias para cada tamanho de problema, para que os resultados sejam representativos. Os dados mostrados nesta tabela consideram tanto instâncias da classe A quanto da classe B. Então, por exemplo, a entrada na primeira linha, que corresponde a $N = 40$ tarefas e $F = 10$ clientes, considera a geração de 20 instâncias, sendo 10 da classe A e 10 da classe B. Ao todo foram geradas 80 instâncias de grande porte.

Tabela 3.16. Conjunto das 80 instâncias de grande porte geradas, considerando classes A e B.

N	**F**	**# Instâncias**	**Total**
40	10	10	20
60	10	10	20
80	10	10	20
100	10	10	20
Total			80

3.5.6.2 Experimento 1 - Análise das regras de dominância na busca local

Foi realizado um experimento com o intuito de verificar como as regras de dominância propostas na Seção 3.4.3.1, utilizada na busca local API_LS, efetivamente conseguiram melhorar a eficiência deste procedimento. Neste experimento, foram utilizadas todas as 80 instâncias de grande porte. Para cada problema teste, uma solução é construída pela heurística CONSTRUCTION e, então, sobre esta solução, é aplicada a busca local API_LS, com e sem as regras de dominância. Este processo é repetido 100 vezes para cada instância. A variável de resposta é o tempo médio gasto nas 100 rodadas. São feitas 30 repetições do experimento.

Seja *Fast* a versão da busca local API_LS utilizando as regras de dominância na busca local e a versão *Normal* aquela que não as utiliza. Para analisar a eficiência

das propriedades propostas, foi realizada uma comparação entre estas duas versões. Foi realizado um *teste-t* para comparação das médias dos dois algoritmos. O nível de confiança utilizado foi de 95.0%. O *valor-P* computado para o teste foi $0{,}00 < 0{,}05$ mostrando que há uma diferença significativa entre os dois algoritmos. O gráfico de caixa (*Box-and-Whisker Plot*), utilizado para avaliar a distribuição empírica dos dados, da Figura 3.8 mostra que a variante *Fast* tem desempenho muito melhor que a variante *Normal*, uma vez que não há sobreposição dos intervalos e a média de tempo é visivelmente menor. A média para a variante *Fast* foi de 27,83 ms enquanto que para *Normal* foi 1451,27 ms.

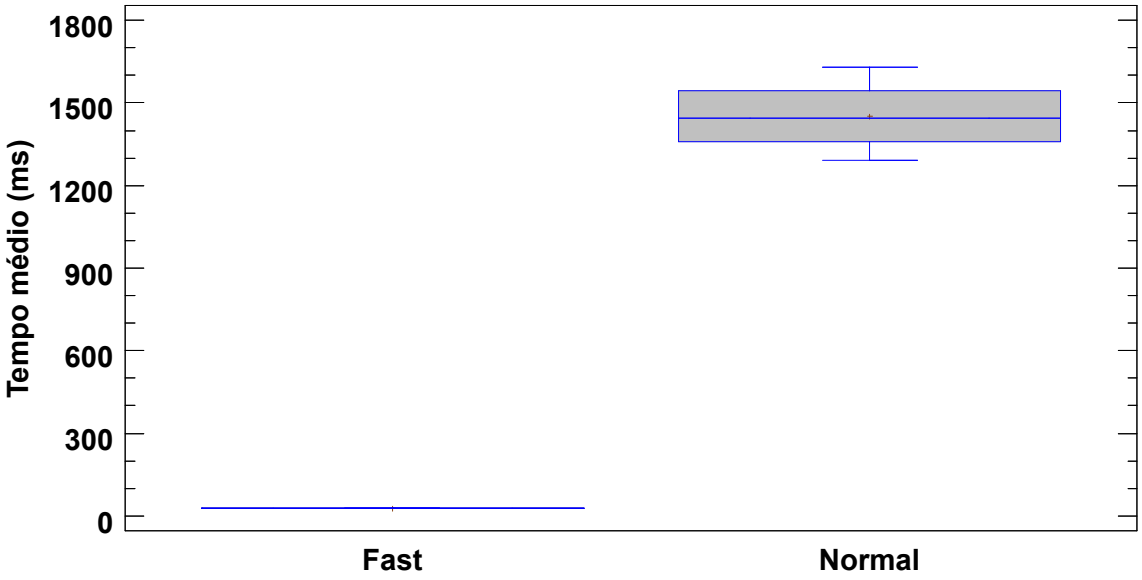


Figura 3.8. Gráfico de caixas para o tempo médio entre as versões *Fast* e *Normal* da busca local API_LS.

Foi calculado também a aceleração média, por tamanho de problema. A aceleração é calculada, para cada problema, como a razão entre o tempo gasto pela versão *Normal* sobre a versão *Fast*. Os resultados são mostrados no gráfico da Figura 3.9. Observe que o ganho em tempo são consideráveis variando deste 67 vezes, para instâncias com 40 tarefas, a quase 93 vezes, para instâncias com 60 tarefas onde houve o maior ganho de tempo. A aceleração média foi de 82,61 vezes. Os resultados deste experimento indicam que é viável a utilização das regras de dominância na busca local API_LS.

3.5.6.3 Experimento 2 - Calibração dos parâmetros do algoritmo ILS

Na calibração dos parâmetros dos algoritmos ILS e IG, para a realização dos experimentos com instâncias de grande porte, foi utilizado um conjunto de 80 instâncias

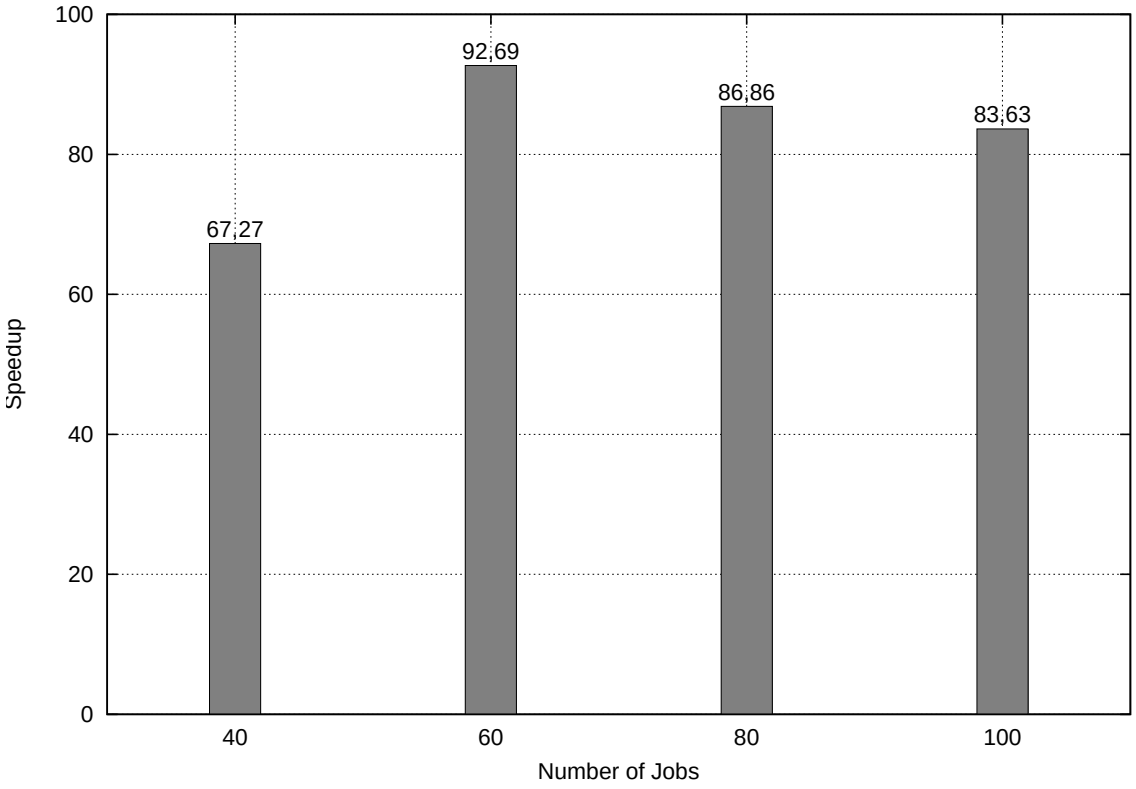


Figura 3.9. Aceleração média *versus* número de tarefas.

geradas independentes e exclusivamente para o ajuste de parâmetros dos algoritmos conforme descrito na Seção 3.5.6.1.

O algoritmo ILS possui 3 parâmetros de entrada: d, β e *numberSwap*. Os valores testados para cada um destes parâmetros estão especificados na Tabela 3.17. Foi utilizado o desenho fatorial completo, o que dá um total de 27 combinações a serem testadas. O algoritmo ILS será executado 5 vezes para cada instância e para cada combinação de parâmetro, de modo que para este experimento tem-se um total de 10800 amostras. Foi utilizado um critério de parada que é baseado em um tempo máximo de CPU igual a $500 \times N$ milissegundos. Os resultados foram analisados através do RPD% (calculado conforme equação (3.30)) e ANOVA paramétrica.

Tabela 3.17. Conjunto de valores testados para a calibração do ILS considerando instâncias de grande porte.

Parâmetro	**Valores testados**	**# Valores**
d	$\{2; 3; 4\}$	3
β	$\{0{,}0; 0{,}1; 0{,}2\}$	3
numberSwap	$\{N; 2N; 3N\}$	3
Total		27

A variável de resposta considerada na ANOVA é o RDP% médio, enquanto que os tratamentos serão as configurações dos parâmetros consideradas na calibração. Foram verificadas as pressuposições da ANOVA, cujos teste estão organizados no Apêndice C.2. O resultado da ANOVA pode ser visto na Tabela 3.18. O *valor de F*, que neste caso é igual a 2331,89, é o quociente entre o quadrado médio dos tratamentos pelo quadrado médio do resíduos. Uma vez que o *valor-P* do teste-F, que é o valor de interesse, é menor que 0,05, há uma diferença estatisticamente significante de uma das configurações para outra, com um nível de confiança de 95%.

Tabela 3.18. Tabela ANOVA para comparação das configurações testadas na calibração do algoritmo ILS, considerando instâncias de grande porte

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	1085,71	26	41,76	**2331,89**	**0,00**
Resíduos	1,93	108	0,02		
Total	1087,65	134			

Para identificar onde estão os contrastes entre os tratamentos foi realizado um teste de Comparações Múltiplas. A Figura 3.10 mostra o gráfico de médias resultante do teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95% para as configurações testadas no algoritmo ILS. Nesta figura é possível ver que existe diferença significativa entre as configurações, pois há diversos intervalos que não se sobrepõe, mesmo embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. A configuração 1 apresentou a melhor média e corresponde aos valores $d = 2$; $\beta = 0{,}0$ e $numberSwap = 2N$.

3.5.6.4 Experimento 3 - Calibração dos parâmetros do algoritmo IG

A calibração dos parâmetros do algoritmo IG se deu de forma semelhante a do ILS. O algoritmo IG possui 3 parâmetros de entrada: d, β e *numberSwap*. Os valores testados para cada um dos parâmetros estão especificados na Tabela 3.19, num total de 27 configurações de IG avaliadas.

O resultado da ANOVA pode ser visto na Tabela 3.20. O *valor de F* neste caso é igual a 11,75 e o *valor-P* do teste-F, que é o valor de interesse, é menor que 0,05, indicando que há uma diferença estatisticamente significante de uma das configurações para outra, com um nível de confiança de 95%. Mais detalhes sobre os pressupostos da ANOVA para este experimento se encontram no Apêndice C.3.

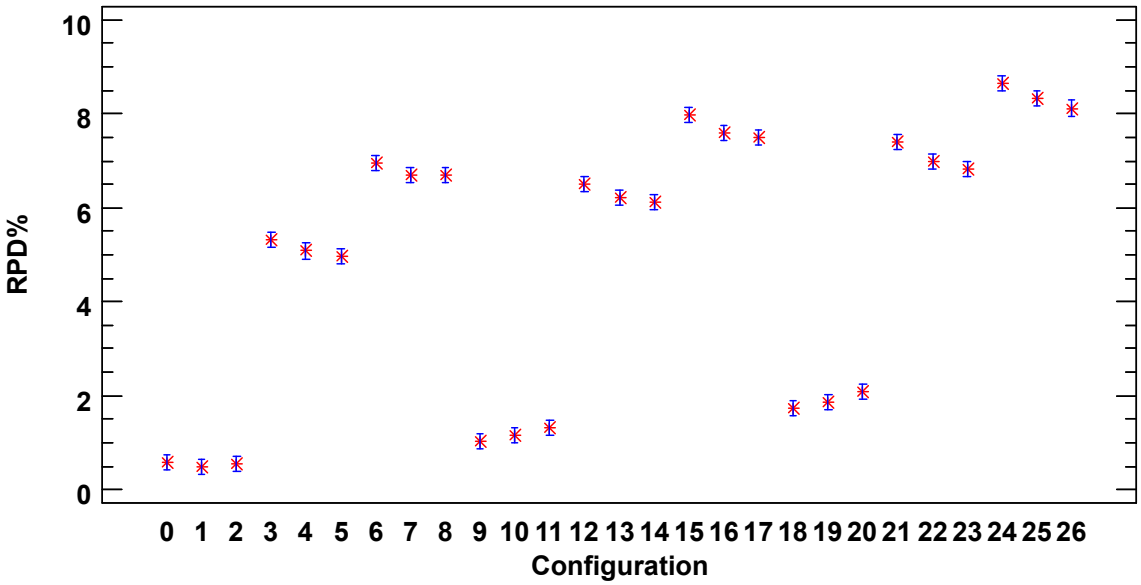


Figura 3.10. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 27 configurações de ILS considerando instâncias de grande porte.

Tabela 3.19. Conjunto de valores testados para a calibração do algoritmo IG considerando instâncias de grande porte.

Parâmetro	**Valores testados**	**# Valores**
d	$\{6; 8; 10\}$	3
β	$\{0{,}0; 0{,}1; 0{,}2\}$	3
numberSwap	$\{N; 2N, 3N\}$	3
Total		27

Tabela 3.20. Tabela ANOVA para comparação das configurações testadas na calibração do algoritmo IG considerando instâncias de grande porte.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	0,57	26	0,02	**11,75**	**0,00**
Resíduos	0,20	108	0,00		
Total	0,78	134			

A Figura 3.11 mostra o gráfico de médias resultante do teste *Tukey* da Diferença Honestamente Significativa HSD com nível de confiança de 95% para as 27 configurações testadas do algoritmo IG, de onde podem ser vistos os contrastes entre os tratamentos. Note que existe diferença significativa entre as configurações, pois há diversos intervalos que não se sobrepõe, mesmo embora não haja uma única configuração que seja estatisticamente melhor que todas as outras. A configuração 23 apresentou a melhor média e corresponde aos valores $d = 10$; $\beta = 0{,}1$ e *numberSwap*

$= 3N$.

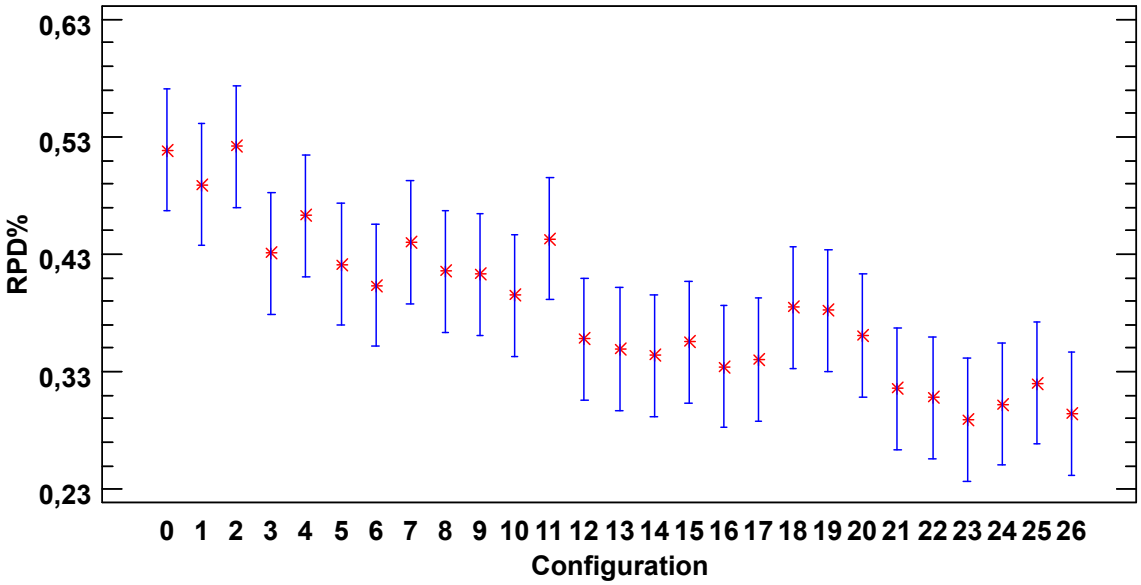


Figura 3.11. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 27 configurações do algoritmo IG considerando instâncias de grande porte.

3.5.6.5 Experimento 4 - Comparação entre os algoritmos ILS e IG

A fim de comparar a eficácia dos algoritmos ILS e IG quanto a qualidade das soluções, considerando somente as 80 instâncias de grande porte, foi conduzido outro experimento. Os algoritmos ILS e IG foram executados com um mesmo critério de parada que é baseado em uma quantidade de tempo de CPU, que foi fixada em $500 \times N$ milissegundos. Para resolver cada instância, cada um dos algoritmos foi executado 30 vezes, sendo que no experimento serão analisadas 4800 amostras. Os resultados obtidos pelos algoritmos ILS e IG foram comparados através do RPD% que é calculado conforme equação (3.30).

A Tabela 3.21 contém os resultados deste experimento organizados quanto ao número de tarefas e clientes. Nesta tabela, a classe de problemas representada por (n, f, *) diz respeito a todos os problemas com o número de tarefas $N = n$ e o número de clientes $F = f$. Cada entrada é o RPD% médio, considerando todas as instâncias de uma dada classe e o valor da função objetivo em todas as 30 execuções de um dado algoritmo. Como pode ser visto na Tabela 3.21 o algoritmo IG possui os menores valores de RPD% para todas as classes de problemas, o que é um indício de que este algoritmo encontra soluções de melhor qualidade quando comparado ao algoritmo ILS. O RPD% médio do algoritmo IG foi de 0,023% enquanto que o RPD% médio do

algoritmo ILS foi de 0,979%. Apesar de apresentarem valores relativamente baixos de RPD%, quanto se olha o valor do custo das soluções encontradas por cada um dos algoritmos nas 4800 amostras esta diferença chega a 6043,50. A média do custo das soluções encontradas pelo algoritmo ILS foi de 520699,61 e pelo algoritmo IG foi de 514656,11. A performance do algoritmo ILS piora quando se aumenta o número de tarefas enquanto o algoritmo IG não apresenta o mesmo comportamento.

Tabela 3.21. Percentual de Desvio Relativo (RPD%) médio considerando somente número de tarefas e clientes.

Classe de problemas	**# Instâncias**	**ILS**	**IG**
(40, 10, *)	20	0,706%	**0,003%**
(60, 10, *)	20	0,851%	**0,068%**
(80, 10, *)	20	1,072%	**0,016%**
(100, 10, *)	20	1,288%	**0,004%**
Média		0,979%	**0,023%**

De forma semelhante é feita uma análise considerando somente as classes A ou B cujos resultados estão dispostos na Tabela 3.22. Novamente, o algoritmo IG encontra as melhores soluções tanto para as classes A ou B. Vale ressaltar que o algoritmo ILS se mostra bem menos competitivo para os problemas da classe A, onde foram alcançados os piores resultados para este algoritmo com RPD% médio de 1,692%.

Tabela 3.22. Percentual de Desvio Relativo (RPD%) médio considerando somente as classes de problemas A e B.

Classe de problemas	**# Instâncias**	**ILS**	**IG**
(*, *, A)	40	1,692%	**0,043%**
(*, *, B)	40	0,267%	**0,068%**
Média		0,979%	**0,023%**

Estes resultados exibidos anteriormente indicam que o algoritmo IG é o melhor algoritmo quanto a qualidade das soluções encontradas. Para verificar se as diferenças observadas no experimento são estatisticamente significantes foi conduzido um *teste-t* para comparação das médias das soluções encontradas pelos dois algoritmos. Este teste verifica se a hipótese nula de que a média das soluções dos dois algoritmos são estatisticamente iguais contra a hipótese alternativa de que as médias das soluções não sejam iguais. O nível de confiança escolhido para a realização do *teste-t* é de 95%. Após a realização do teste, o *valor-P* computado foi de $0{,}00 < 0{,}05$ de modo que se rejeite a hipótese nula em favor da alternativa. Ou seja, as diferenças observadas são estatisticamente significantes.

Através do gráfico de caixas (*Box and Whisker Plot*), mostrado na Figura 3.12, que é utilizado para avaliar a distribuição empírica dos dados, é possível ver que o algoritmo IG tem desempenho muito melhor que o algoritmo ILS uma vez que não há sobreposição dos intervalos e a média RPD% é visivelmente menor.

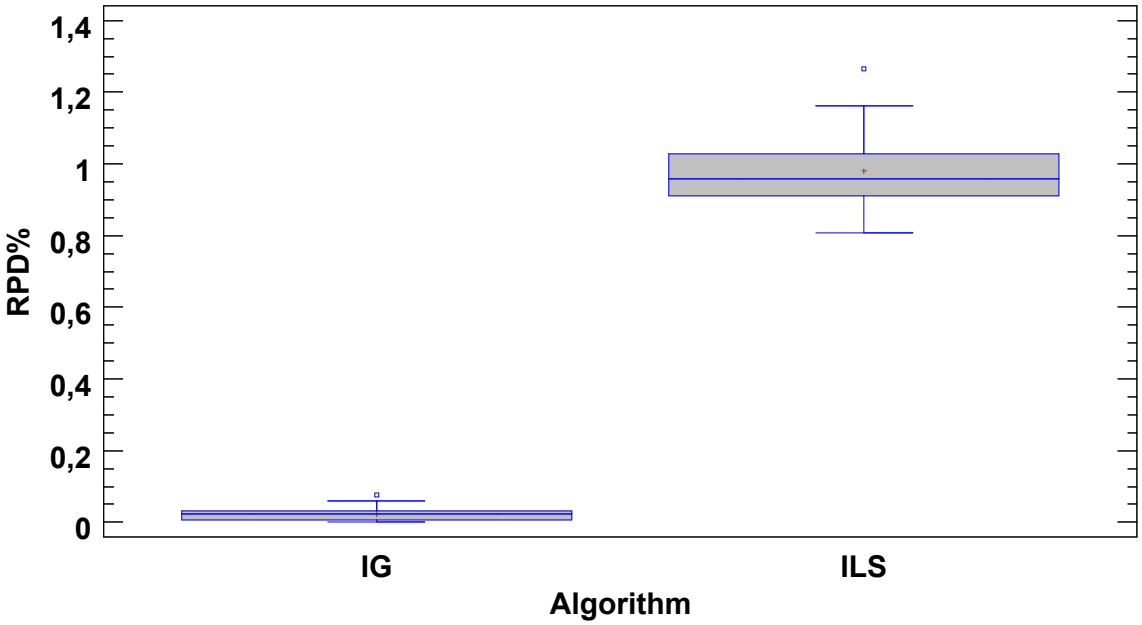


Figura 3.12. Gráfico de caixas para comparação das médias do RPD% entre os algoritmos ILS e IG considerando somente instâncias de grande porte.

3.5.6.6 Experimento 5 - Análise de tempo

Os experimentos conduzidos anteriormente foram feitos com o foco na qualidade das soluções com respeito ao valor da função objetivo. No entanto, é importante considerar também o desempenho dos algoritmos quanto ao tempo de execução.

Para este fim, primeiramente, foi realizado um experimento variando-se o tempo de execução dos algoritmos. Lembre que os algoritmos ILS e IG foram executados com um mesmo critério de parada que é baseado em uma quantidade de tempo de CPU, que foi fixada em $t \times N$ milissegundos, onde t é uma constante e havia sido fixada com o valor 500. É interessante analisar o comportamento destes dois algoritmos quando há valores de t menores e maiores que 500, principalmente por causa do componente construtivo do algoritmo IG que pode consumir muito tempo de processamento de modo que exista a possibilidade de o algoritmo ILS se comportar melhor quando há valores menores de t.

Neste experimento, para se resolver cada instância, cada um dos algoritmos foi executado 30 vezes, com valores de $t \in \{250; 500; 1000\}$, sendo que no experimento

serão analisadas 14400 amostras. Os resultados obtidos pelos algoritmos ILS e IG foram comparados através do RPD% que é calculado conforme equação (3.30).

A Figura 3.13 mostra o gráfico de interações entre o tempo de CPU, utilizado como critério de parada, e o RPD% médio obtido por cada um dos algoritmos. É possível observar que o algoritmo IG é significativamente melhor que o algoritmo ILS para todos os três valores de t testados, inclusive para $t = 250$. Além disso, é possível verificar que tanto o algoritmo ILS quanto o algoritmo IG encontram soluções significativamente melhores com maiores tempos de execução, o que é um indicativo de que os procedimentos de melhorias efetivamente continuam funcionando.

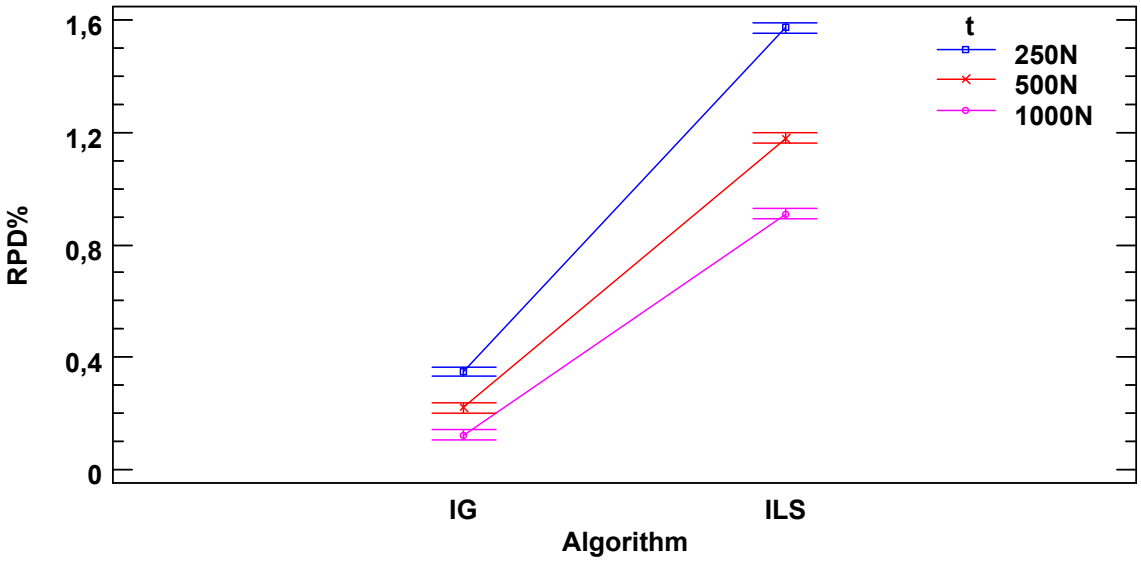


Figura 3.13. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a interação entre o critério de parada e o RPD% médio dos algoritmos ILS e IG considerando instâncias de grande porte.

Foi realizado um outro experimento com o intuito de analisar a convergência das heurísticas ILS e IG propostas. Neste experimento foi registrado o valor da função objetivo obtida por cada método durante a sua execução. Foi utilizada apenas uma instância da classe A com 100 tarefas: a instância "100-10-A-0". É importante destacar que este experimento não possui validação estatística e por isso não há a necessidade de utilização de muitas instâncias, uma vez que o que se quer analisar é apenas uma tendência dos dois métodos ao convergir para uma determinada solução.

Cada um dos algoritmos foi executado 30 vezes, sendo o critério de parada um tempo de CPU igual a 10 horas. A melhor solução obtida para a instância, considerando-se todas as 60 observações, é utilizada como referência, já que obter a

solução ótima para problemas com 100 tarefas é inviável neste mesmo intervalo de tempo, como demonstrado em experimentos anteriores.

Nos gráficos das Figuras 3.14 e 3.15 são apresentados os resultados obtidos para a instância "100-10-A-0". A solução utilizada como referência (destacada no gráfico como "*Best Solution*") possui custo de 1029216 e foi encontrada pelo algoritmo IG. A melhor solução encontrada pelo algoritmo ILS em 10 horas de execução foi 1035559 que está 0,62% acima da melhor solução conhecida. Na Figura 3.14 é possível observar que o algoritmo IG parece convergir mais rapidamente para a solução referência uma vez que a curva associada a este algoritmo fica abaixo daquela apresentada pelo algoritmo ILS logo nos primeiros segundos de execução.

Além disso, em 1 hora de execução o algoritmo IG fica mais próximo da melhor solução conhecida encontrando uma solução com custo 1032878, solução esta encontrada após 705 segundos e que está 0,36% acima da solução referência. Em 1 hora de execução a melhor solução encontrada pelo ILS possui valor de 1040894, encontrada após 2584 segundos e que está 1,13% acima da solução referência. Observe que neste experimento em 1 hora de execução o algoritmo IG encontrou uma solução melhor que o algoritmo ILS em 10 horas de execução.

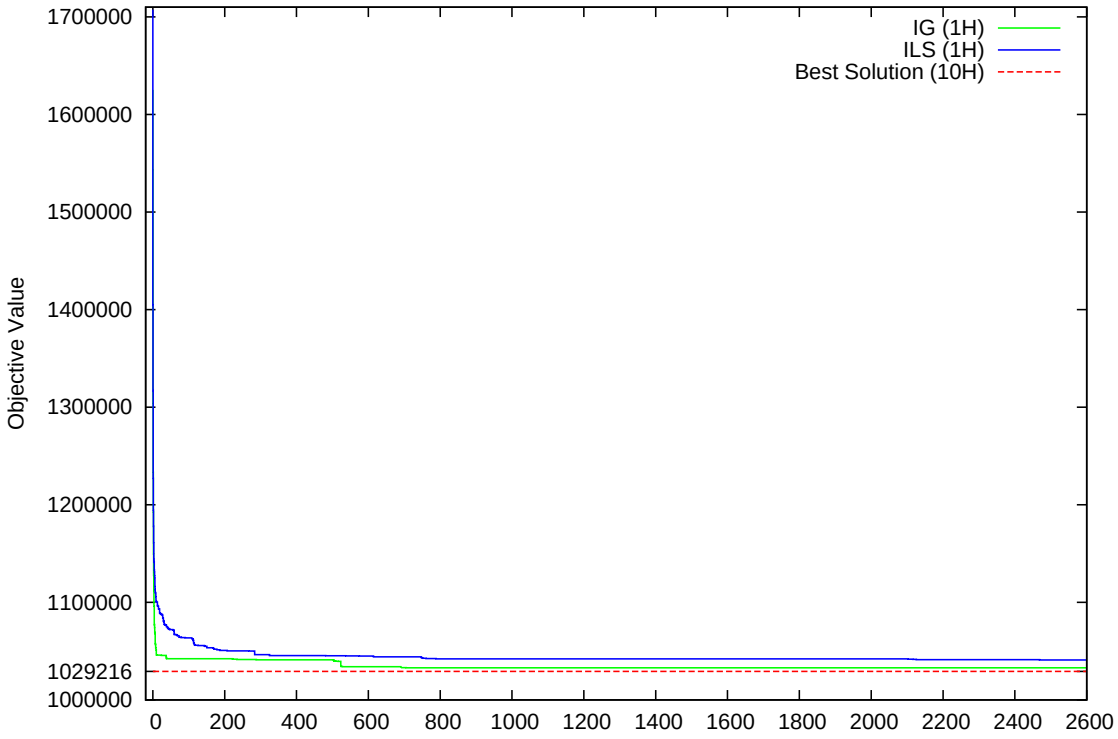


Figura 3.14. Convergência dos algoritmos ILS e IG para a instância "100-10-A-0". Intervalo de tempo exibido no gráfico entre 0 e 2600 segundos.

Na Figura 3.15 é apresentado os mesmos resultados, porém mostrando somente os dois primeiros segundos de execução. Nesta figura observa-se que nos dois primeiros segundos o algoritmo ILS é competitivo com o IG, tendo inclusive em diversos intervalos de tempos soluções de melhor qualidade que o IG, já que sua curva fica abaixo daquela correspondente ao algoritmo IG. No entanto, a partir de 1,4 segundos até o final de 1 hora de execução, a heurística IG se sobressai. Este comportamento pode ser devido ao fato de as iterações do ILS serem mais rápidas que as do IG, principalmente por causa do componente reconstrutor na heurística IG. Por ter iterações mais rápidas o ILS consegue nos primeiros segundos diversas melhorias.

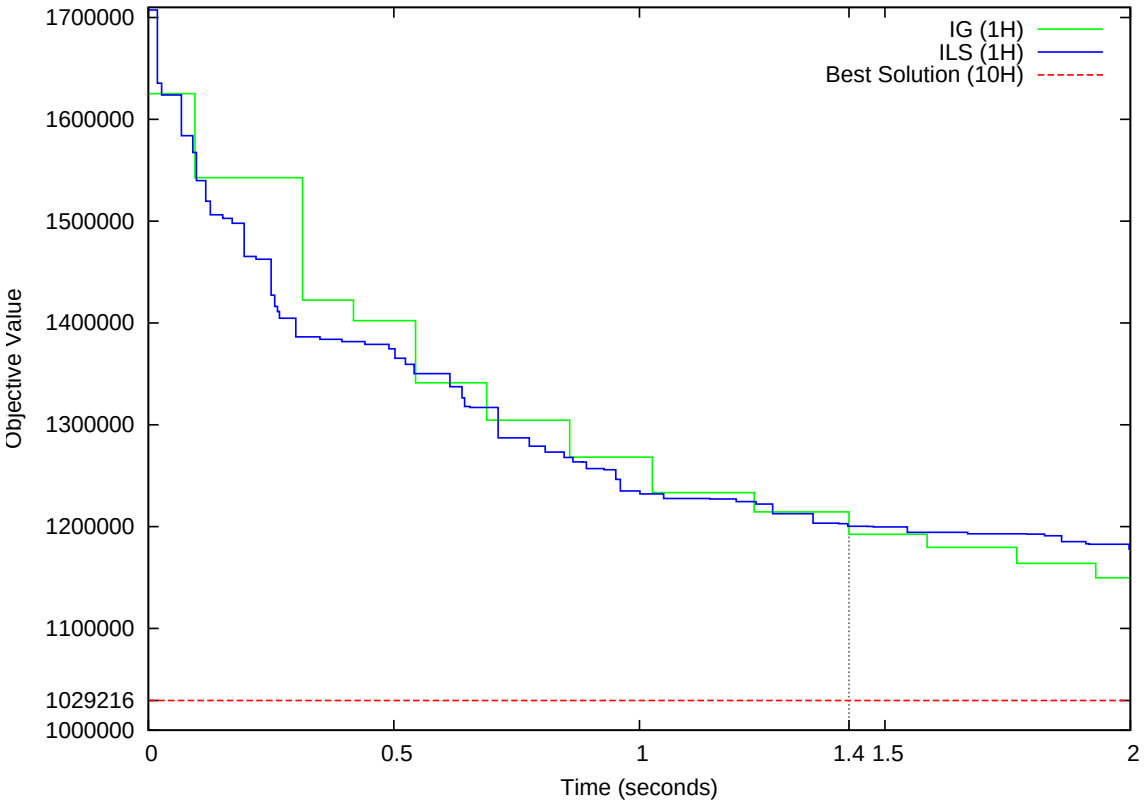


Figura 3.15. Convergência dos algoritmos ILS e IG para a instância "100-10-A-0". Intervalo de tempo exibido no gráfico vai até 2 segundos.

Como o tempo de CPU consumido pelos algoritmos IG e ILS em cada iteração são muito diferentes, foi realizado outro experimento onde o critério de parada dos algoritmos foi a obtenção de uma solução cujo valor da função objetivo seja menor ou igual a um valor alvo previamente predefinido. Este é um tipo de análise de desempenho empírica que se utiliza de testes de probabilidade empírica juntamente com *time-to-target plots*. Estes procedimentos de análise tem sido utilizados no

desenvolvimento de algoritmos e como estratégia de comparações de implementações (HADDAD, 2012). De forma geral, cada algoritmo é aplicado $n = 100$ vezes e sempre que o valor alvo é alcançado o tempo de execução (em segundos) é registrado e o algoritmo é interrompido. Os tempos de execução são, então, ordenados de forma não decrescente. Para cada tempo de execução t_i, registrado por um determinado algoritmo, é calculada a probabilidade cumulativa $p_i = \frac{(i-0,5)}{n}, \forall i = 1, 2, \cdots, n$.

Para a realização deste experimento foi utilizada a mesma instância que no experimento anterior (a instância "100-10-A-0") que analisou a convergência entre os métodos. A melhor solução conhecida para esta instância foi encontrada somente pelo algoritmo IG e possui custo 1029216. Como o algoritmo ILS não encontrou em nenhuma das 30 repetições este valor, será utilizado um valor alvo um pouco acima da melhor solução conhecida. Nos experimentos serão considerados dois tipos de alvos: um alvo "fácil" que corresponde ao valor 10% acima da melhor solução conhecida e um alvo "difícil" que corresponde ao valor 5% acima da melhor solução conhecida.

O gráfico *time-to-target plots* gerado é apresentado na Figura 3.16 e corresponde a comparação entre os algoritmos ILS e IG para o alvo "fácil". Observe que as curvas foram sobrepostas para fique mais fácil a visualização das curvas de probabilidades empíricas. Ao analisar o gráfico, é fácil observar que a implementação IG tem uma maior probabilidade de encontrar a solução alvo mais rapidamente que o ILS. Por exemplo, a probabilidade de o algoritmo IG encontrar uma solução tão boa quanto um valor alvo, para a instância "100-10-A-0" e considerando o alvo "fácil", abaixo dos 3 segundos é de cerca de 92,57% enquanto que neste mesmo tempo a probabilidade do algoritmo ILS é de apenas 71,78%. A probabilidade considerando o tempo abaixo de 2 segundos para o IG foi de 13,37% e para o ILS foi de 6,44%. Para alcançar a solução alvo com 98,51% de probabilidade o IG precisa de 3,49 segundos enquanto que o algoritmo ILS precisa de 4,59 segundos para alcançar o alvo com a mesma probabilidade.

A mesma análise foi realizada considerando-se o alvo "difícil" e instância "100-10-A-0", sendo que os resultados são apresentados na Figura 3.17. Aqui as diferenças entre as duas implementações ficam ainda mais acentuadas mostrando claramente que a implementação IG possui melhor performance quanto a encontrar uma solução alvo mais rapidamente. A probabilidade de o algoritmo IG encontrar uma solução tão boa quanto um valor alvo, para a instância "100-10-A-0" e considerando o alvo "difícil", abaixo dos 5,6 segundos é de cerca de 98,51% enquanto que neste mesmo tempo o algoritmo ILS não chegou ao valor alvo nenhuma vez. Para alcançar esta mesma probabilidade o algoritmo ILS precisa de pelo menos 41,32 segundos. O algo-

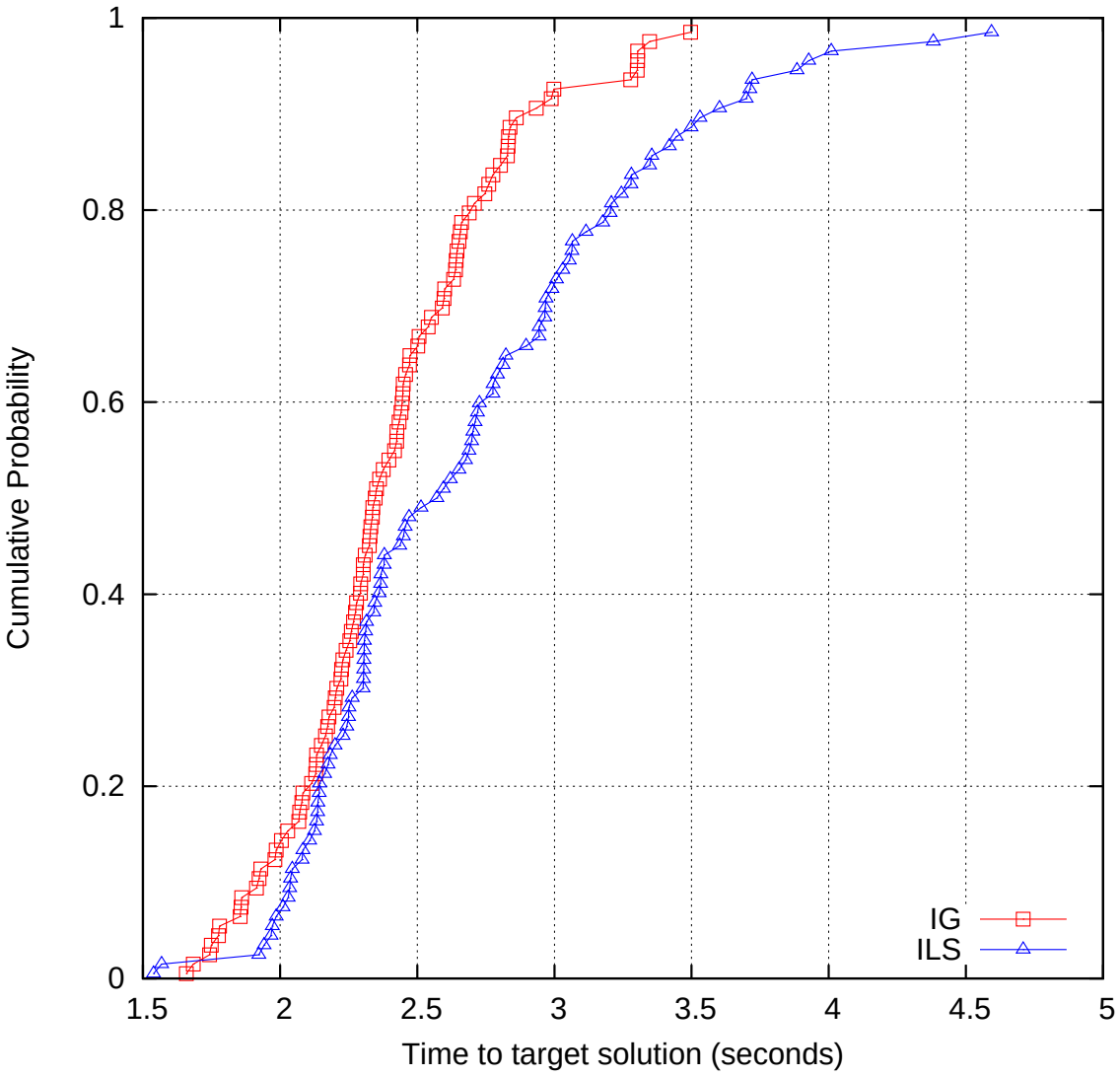


Figura 3.16. Gráfico de distribuição de probabilidades cumulativas dos algoritmos ILS e IG considerando o alvo "fácil"- instância "100-10-A-0".

ritmo ILS alcança o alvo com tempo abaixo de 20 segundos com uma probabilidade de 90,59%.

No entanto, é necessário validar estes resultados através de métodos estatísticos. Para este fim, foi utilizada a ferramenta numérica, proposta nos trabalhos realizados por Ribeiro & Rosseti (2009) e Ribeiro et al. (2012), que compara qualquer par de algoritmos de busca local estocásticos. Esta ferramenta calcula a probabilidade de que um determinado algoritmo A_1 encontre uma solução tão boa quanto uma solução alvo em um tempo menor que outro algoritmo A_2. Denota-se por X_1 e X_2 a variável aleatória contínua representando o tempo necessário para os algoritmos A_1 e A_2, respectivamente, encontrarem uma solução tão boa quanto um alvo

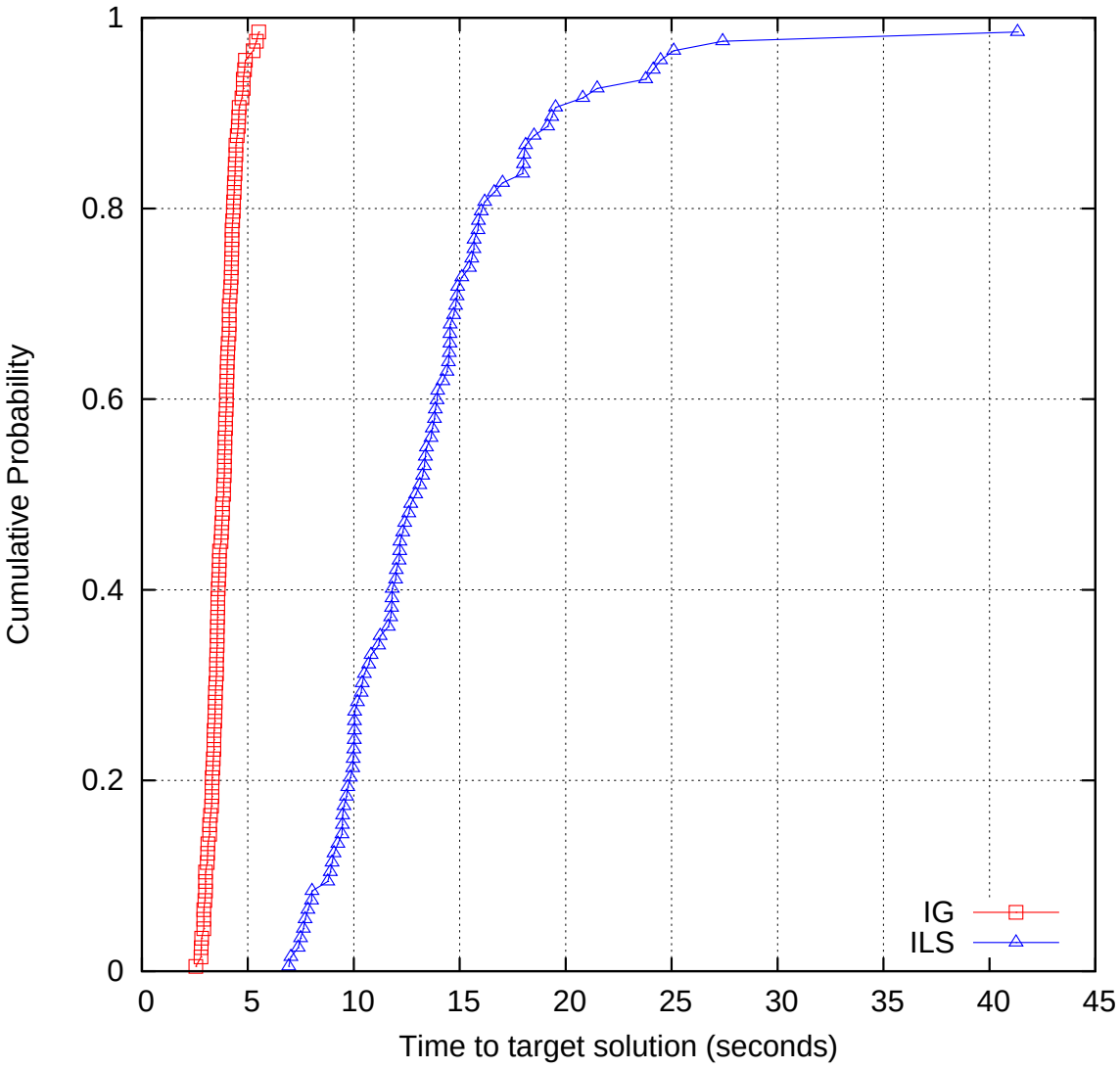


Figura 3.17. Gráfico de distribuição de probabilidades cumulativas dos algoritmos ILS e IG considerando o alvo "difícil"- instância "100-10-A-0".

para um determinado problema teste. O objetivo é determinar a probabilidade, $Pr(X_1 \leq X_2)$, que X_1 tenha um valor menor ou igual a $X2$, o que, em outras palavras, significa que o algoritmo A_1 possui performance melhor que A_2.

Seja A_1 o algoritmo IG e A_2 o algoritmo ILS. Primeiramente, é feita a validação com relação ao alvo "fácil" e instância "100-10-A-0". A probabilidade calculada de o algoritmo IG encontrar uma solução tão boa quanto o alvo em um tempo inferior ao algoritmo ILS foi de $Pr(A_1 \leq A_2) = 62{,}13\%$ com um erro de 0,05%. Para o alvo "difícil" e instância "100-10-A-0" a probabilidade calculada foi de $Pr(A_1 \leq A_2) = 99{,}96\%$ com um erro de 0,09%.

3.6 Conclusões

A proposta deste estudo foi examinar estratégias para um problema de programação de produção, em uma máquina, que envolve tanto o sequenciamento de tarefas quanto custos de transporte associados a entrega de lotes. O problema também considera tempos de liberação das tarefas e prioridades. O problema $1|r_{ij}|\sum w_{ij}F_{ij} + \sum \delta_i d_i$ é um problema complexo de otimização combinatória com ampla aplicação nas indústrias principalmente quando se considera cadeia de fornecimento com dois estágios. Em virtude da natureza $\mathcal{NP}$-Difícil deste tipo de problema, dois algoritmos heurísticos baseados na metaheurística *Iterated Local Search* (ILS) e *Iterated Greedy* (IG), que mantém tanto simplicidade quando generalidade, são propostos. Além destas duas heurísticas, foi proposto também um modelo matemático de Programação Linear Inteira Mista (MILP). Os parâmetros dos algoritmos heurísticos foram analisados e determinados através da metodologia Desenho de Experimentos e pela Análise de Variância e outros testes estatísticos.

Para avaliar a aplicabilidade das heurísticas ILS e IG propostas, seus desempenhos foram comparados com um algoritmo *Branch and Bound* (B&B) disponível na literatura em um grande conjunto de problemas testes. São mostrados resultados também do modelo matemático que pode resolver problemas com até 10 tarefas e 5 clientes. Os resultados computacionais claramente indicam que as heurísticas propostas são mais efetivas que o B&B e que o modelo matemático para diversas classes de problemas. Desvios muito baixos foram alcançados pelos algoritmos propostos. Além disso, quando se compara à soluções ótimas, provadas através do MILP, verifica-se que dos 112 ótimos encontrados a heurística ILS encontrou o mesmo valor em 99 casos, o que é um indício de que a heurística realmente encontra boas soluções.

Neste trabalho, também foram propostas algumas propriedades estruturais que podem acelerar em média em até 92 vezes o procedimento de busca local utilizado. As heurísticas ILS e IG propostas foram comparadas entre si sendo os resultados validados estatisticamente. Os resultados claramente demostram a superioridade da heurística IG sobre a heurística ILS tanto com relação a qualidade das soluções quanto ao tempo de execução. A análise quanto ao tempo de execução e convergência dos algoritmos foram realizadas através de testes de probabilidades empíricas, indicando superioridade da versão IG sobre a versão ILS. Foram demonstrados resultados para problemas testes com até 100 tarefas e 10 clientes que são relativamente maiores que trabalhos anteriores da literatura. Além disso, as instâncias consideradas neste trabalho são mais genéricas que as utilizadas por outros autores por não

considerar algumas restrições que os problemas testes devem obedecer para que seus algoritmos possam ser aplicados.

Como trabalhos futuros, testar outras vizinhanças de buscas locais, como busca local por inserção, principalmente na heurística ILS pode ser adequado, mas principalmente elaborar outras estratégias de perturbação na heurística ILS ou outros procedimentos gulosos mais eficientes na fase de reconstrução da heurística IG podem surtir melhores resultados. Avaliar a aplicabilidade de procedimentos de controle automático de perturbação e da técnica *Path Relinking* também podem ser tentados. Finalmente, hibridizar o modelo matemático com a heurística IG pode melhorar os resultados do modelo já que, como demonstrado, a heurística IG encontra soluções de melhor qualidade que ILS.

3.7 Publicações

O trabalho desta dissertação gerou as seguintes publicações:

IEEE The International Conference on Artificial Intelligence (IEEE ICAI, 2013)

- *Título:* An ILS Heuristic for the Single Machine Scheduling Problem with Sequence Dependent Family Setup Times to Minimize Total Tardiness
- *Autores:* Jacob, V. V., Arroyo; J. E. C.; Villadiego, H. M. M.

Simpósio Brasileiro de Pesquisa Operacional (XLV SBPO, 2013)

- *Título:* Heurística Busca Local Iterada para o sequenciamento de tarefas em uma máquina com tempos de preparação dependentes da família
- *Autores:* Jacob, V. V., Arroyo; J. E. C.; Santos, A. G.

Aceito para publicação no Simpósio Brasileiro de Pesquisa Operacional (XLVI SBPO, 2014)

- *Título:* Uma heurística ILS para a minimização do fluxo total ponderado e custos de entrega no sequenciamento de tarefas em uma máquina com formação de lotes
- *Autores:* Jacob, V. V., Arroyo; J. E. C.; Santos, A. G.

Referências Bibliográficas

Aiex, R. M.; Resende, M. G. & Ribeiro, C. C. (2002). Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, 8(3):343--373.

Akrout, H.; Jarboui, B.; Siarry, P. & Rebaï, A. (2012). A grasp based on de to solve single machine scheduling problem with sdst. *Computational Optimization and Applications*, 51(1):411--435.

Allahverdi, A.; Gupta, J. N. & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219--239.

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

Armentano, V. A. & Mazzini, R. (2000). A genetic algorithm for scheduling on a single machine with set-up times and due dates. *Production Planning & Control*, 11(7):713--720.

Arroyo, J. E. C.; Nunes, G. V. P. & Kamke, E. H. (2009). Iterative local search heuristic for the single machine scheduling problem with sequence dependent setup times and due dates. Em *Hybrid Intelligent Systems, 2009. HIS'09. Ninth International Conference on*, volume 1, pp. 505--510. IEEE.

Baker, K. R. & Magazine, M. J. (2000). Minimizing maximum lateness with job families. *European Journal of Operational Research*, 127(1):126 – 139.

Belouadah, H.; Posner, M. & Potts, C. (1992). Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics*, 36(3):213--231.

Bianco, L. & Ricciardelli, S. (1982). Scheduling of a single machine to minimize total weighted completion time subject to release dates. *Naval Research Logistics Quarterly*, 29(1):151--167.

Blum, C. & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268--308.

Byong-Hun, A. & Jae-Ho, H. (1990). Single facility multi-class job scheduling. *Computers & Operations Research*, 17(3):265--272.

Chandra, R. (1979). On n/1/f dynamic deterministic problems. *Naval Research Logistics Quarterly*, 26(3):537--544.

Chantaravarapan, S.; Gupta, J. N. D. & Smith, M. L. (2003). A hybrid genetic algorithm for minimizing total tardiness on a single machine with family setups. *POMS meeting*.

Cheng, T.; Gordon, V. S. & Kovalyov, M. Y. (1996). Single machine scheduling with batch deliveries. *European Journal of Operational Research*, 94(2):277--283.

Chu, C. (1992). A branch-and-bound algorithm to minimize total flow time with unequal release dates. *Naval Research Logistics (NRL)*, 39(6):859--875.

Dessouky, M. & Deogun, J. (1981). Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM Journal on Computing*, 10(1):192--202.

Dong, X.; Huang, H. & Chen, P. (2009). An iterated local search algorithm for the permutation flow-shop problem with total flowtime criterion. *Computers and Operations Research*, 36:1664--1669.

Du, J. & Leung, J. Y. (1990). Minimizing total tardiness on one machine is np-hard. *Math. Oper. Res.*, 15(3):483--495.

Feo, T. A.; Resende, M. G. & Smith, S. H. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):860--878.

França, P. M.; Mendes, A. & Moscato, P. (2001). A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132(1):224--242.

Gagné, C.; Price, W. & Gravel, M. (2002). Comparing an aco algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *Journal of the Operational Research Society*, pp. 895--906.

Ganeshan, R. & Harrison, T. P. (1995). An introduction to supply chain management. *Department of Management Science and Information Systems, Penn State University*.

Glover, F. (1996). Tabu search and adaptive memory programing - advances, applications and challenges. *Interfaces in Computer Science and Operations Research*, pp. 1--75.

Graham, R.; Lawler, E.; Lenstra, J. & Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287 – 326.

Gupta, S. R. & Smith, J. S. (2006). Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, 175(2):722--739.

Haddad, H.; Ghanbari, P. & Moghaddam, A. (2012). A new mathematical model for single machine batch scheduling problem for minimizing maximum lateness with deteriorating jobs. *International Journal of Industrial Engineering*, 3:253–264.

Haddad, M. N. (2012). Algoritmos heurísticos híbridos para o problema de sequenciamento em máquinas não-relacionadas com tempos de preparação dependentes da sequencia. Master's thesis, UFOP.

Hall, N. & Potts, C. (2003). Supply chain scheduling: Batching and delivery. *Operations Research*, 51(4):566--584.

Hamidinia, A.; Khakabimamaghani, S.; Mazdeh, M. & Jafari, M. (2012). A genetic algorithm for minimizing total tardiness/earliness of weighted jobs in a batched delivery system. *Computers & Industrial Engineering*, 62(1):29--38.

Hariri, A. & Potts, C. (1983). An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, 5(1):99--109.

Ji, M.; He, Y. & Cheng, T. (2007). Batch delivery scheduling with batch delivery cost on a single machine. *European journal of operational research*, 176(2):745--755.

Jin, F.; Gupta, J. N. D.; Song, S. & Wu, C. (2010). Single machine scheduling with sequence-dependent family setups to minimize maximum lateness. *JORS*, 61(7):1181–1189.

Kacem, D. (2006). Lower bounds for tardiness minimization on a single machine with family setup times. Em *Computational Engineering in Systems Applications, IMACS Multiconference on*, volume 1, pp. 1034 –1039.

Kahlbacher, H. G. & Cheng, T. E. (1993). Parallel machine scheduling to minimize costs for earliness and number of tardy jobs. *Discrete Applied Mathematics*, 47(2):139--164.

Karabati, S. & Akkan, C. (2005). Minimizing sum of completion times on a single machine with sequence-dependent family setup times. *Journal of the Operational Research Society*, 57(3):271--280.

Laguna, M. & Martí, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44--52.

Lee, C.-Y. & Chen, Z.-L. (2001). Machine scheduling with transportation considerations. *Journal of Scheduling*, 4(3):24.

Lenstra, J.; Rinnooy Kan, A. & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343--362.

Liao, C.-J. & Juan, H.-C. (2007). An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research*, 34(7):1899--1909.

Lourenço, H. R. (1995). Job-shop scheduling: computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347--364.

Lourenço, H. R.; Martin, O. C. & Stützle, T. (2003). *Handbook of Metaheuristics*, chapter Iterated local search, pp. 321--353. F. Glover Eds. Kluwer Academic.

Lourenço, H. R.; Martin, O. C. & Stützle, T. (2010). Iterated local search: Framework and applications. Em *Handbook of Metaheuristics*, chapter 12, pp. 363–398. Springer Publishing Company, Incorporated, 2nd edição.

Lustosa, L.; Mesquita, M. & Oliveira, R. (2008). *Planejamento e controle da produção*. Elsevier Brazil.

Mahdavi Mazdeh, M.; Sarhadi, M. & Hindi, K. S. (2008). A branch-and-bound algorithm for single-machine scheduling with batch delivery and job release times. *Computers & Operations Research*, 35(4):1099--1111.

Martin, O. & Otto, S. W. (1996). Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57--75.

Mathirajan, M. & Sivakumar, A. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9):990--1001.

Mazdeha, M.; Esfahania, A.; Sakkakia, S. & Pileroodb, A. (2012). Single-machine batch scheduling minimizing weighted flow times and delivery costs with job release times. *International Journal of Industrial Engineering*, 3.

Mazdeha, M.; Hamidiniaa, A. & Karamouziana, A. (2011). A mathematical model for weighted tardy jobs scheduling problem with a batched delivery system. *International Journal of Industrial Engineering Computations*, 2:491--498.

Montgomery, D. C. (2006). *Design and Analysis of Experiments*. John Wiley & Sons.

Nawaz, M.; Jr, E. E. E. & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91 – 95.

Pan, J. C.-H. & Su, C.-S. (1997). Single machine scheduling with due dates and class setups. *Journal of the Chinese Institute of Engineers*, 20(5):561--572.

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.

Pinedo, M. (2012). *Scheduling: theory, algorithms, and systems*. Springer.

Potts, C. & Kovalyov, M. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, 120(2):228--249.

Potts, C. N. & Van Wassenhove, L. N. (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, pp. 395--406.

Ragatz, G. L. (1993). A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. Em *Proceedings: twenty-fourth annual meeting of the Decision Sciences Institute*, pp. 1375--1377. John Wiley & Sons, Inc.

Rasti-Barzoki, M.; Hejazi, S. R. & Mazdeh, M. M. (2012). A branch and bound algorithm to minimize the total weighed number of tardy jobs and delivery costs. *Applied Mathematical Modelling*, 1(0):–.

Rego, M. F.; Souza, M. J. F. & Arroyo, J. E. C. (2012). Multi-objective algorithms for the single machine scheduling problems with setup time dependent on the sequence and the job family. Em *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, pp. 1--8. IEEE.

Resende, M. G. C. & Ribeiro, C. C. (2010). *Handbook of Metaheuristics*, chapter Greedy randomized adaptive search procedures: Advances, hybridizations, and applications, pp. 219--249. M. Gendreau and J.Y. Potvin Eds. Springer-Verlag, 2nd Edition.

Ribeiro, C. C. & Rosseti, I. (2009). Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. In Proceeding of the VIII Metaheuristics International Conference, Hamburg.

Ribeiro, C. C.; Rosseti, I. & Vallejos, R. (2012). Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54(2):405--429.

Rubin, P. A. & Ragatz, G. L. (1995). Scheduling in a sequence dependent setup environment with genetic search. *Computers & Operations Research*, 22(1):85--99.

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

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.

Schaller, J. (2007). Scheduling on a single machine with family setups to minimize total tardiness. *International Journal of Production Economics*, 105(2):329--344.

Schaller, J. E. & Gupta, J. N. (2008). Single machine scheduling with family setups to minimize total earliness and tardiness. *European Journal of Operational Research*, 187(3):1050 – 1068.

Selvarajah, E.; Steiner, G. & Zhang, R. (2013). Single machine batch scheduling with release times and delivery costs. *Journal of Scheduling*, 16(1):69--79.

Sewell, E. C.; Sauppe, J. J.; Morrison, D. R.; Jacobson, S. H. & Kao, G. K. (2012). A bb&r algorithm for minimizing total tardiness on a single machine with sequence dependent setup times. *Journal of Global Optimization*, 54(4):791--812.

Shen, W.; Wang, L. & 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*, 36(4):563--577.

Sioud, A.; Gravel, M. & Gagné, C. (2012). A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 39(10):2415--2424.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59--66.

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

Tan, K. & Narasimhan, R. (1997). Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. *Omega*, 25(6):619--634.

Tan, K.-C.; Narasimhan, R.; Rubin, P. A. & Ragatz, G. L. (2000). A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. *Omega*, 28(3):313--326.

Tanaka, S. & Araki, M. (2012). An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. *Computers & Operations Research*.

Vallada, E.; Ruiz, R. & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Comput. Oper. Res.*, 35(4):1350--1373.

Van der Veen, J. A.; Woeginger, G. J. & Zhang, S. (1998). Sequencing jobs that require common resources on a single machine: a solvable case of the tsp. *Mathematical programming*, 82(1-2):235--254.

Ying, K.-C.; Lin, S.-W. & Huang, C.-Y. (2009). Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 36(3):7087--7092.

Apêndice A

Modelo matemático de Chantaravarapan et al. (2003)

Por questão de completude do trabalho, neste apêndice estão transcritos o modelo matemático apresentado no trabalho de Chantaravarapan et al. (2003). Optou-se por colocar o modelo no apêndice porque as notações utilizadas por Chantaravarapan et al. e a utilizada no presente trabalho não são exatamente iguais. Segue o modelo matemático mencionado.

Notations :

$$X_{ijk} = \begin{cases} 1, & \text{if job } j \text{ from family } i \text{ is placed in position } k \\ 0, & \text{otherwise} \end{cases}$$

$$Y_{iqk} = \begin{cases} 1, & \text{if } s_{iq} \text{ is needed before a job at position } k \\ 0, & \text{otherwise} \end{cases}$$

$$Y_{0qk} = \begin{cases} 1, & \text{if } s_{0q} \text{ is needed before a job before position 1} \\ 0, & \text{otherwise} \end{cases}$$

f = number of families.

n_i = number of jobs in family i.

n = total number of jobs $= n_1 + n_2 + \ldots + n_f$.

d_{ij} = due date of the j^{th} job in family i.

p_{ij} = processing time of the j^{th} job in family i.

C_k = completion time of the job at position k.

T_k = tardiness of the job at position k.

s_{iq} = sequence dependent setup time of preceding family i and following family q.

s_{0q} = setup time of family q before the first position.

Formulation

Objective Function $\quad Min \quad Z = \sum_{k=1}^{n} T_k$ (A.1)

Subject to:

$$\sum_{i=1}^{f}\sum_{j=1}^{n_i} X_{ijk} = 1 \qquad k = 1, 2, \ldots, n \quad \text{(A.2)}$$

$$\sum_{k=1}^{n} X_{ijk} = 1 \qquad j = 1, 2, \ldots, n_i, i = 1, 2, \ldots, f \quad \text{(A.3)}$$

$$\sum_{j=1}^{n_q} X_{qj1} = Y_{0q1} \qquad q = 1, 2, \ldots, f \quad \text{(A.4)}$$

$$\sum_{j=1}^{n_q} X_{qjk} + \sum_{j=1}^{n_i} X_{ij(k-1)} - Y_{iqk} \leq 1 \qquad k = 2, 3, \ldots, n, i = 1, 2, \ldots, f, q = 1, 2, \ldots, f, i \neq q \quad \text{(A.5)}$$

$$C_1 = \sum_{q=1}^{f} s_{0q} Y_{0q1} + \sum_{q=1}^{f}\sum_{j=1}^{n_q} p_{qj} X_{qj1} \quad \text{(A.6)}$$

$$C_k = C_{k-1} + \sum_{q=1}^{f}\sum_{i=1}^{f} s_{iq} Y_{iqk} + \sum_{\mathbf{q=1}}^{\mathbf{f}}\sum_{\mathbf{j=1}}^{\mathbf{n_q}} \mathbf{p_{qj} X_{qjk}} \qquad k = 2, 3, \ldots, n \quad \text{(A.7)}$$

$$C_k - \sum_{i=1}^{f}\sum_{j=1}^{n_i} d_{ij} X_{ijk} \leq T_k \qquad k = 1, 2, \ldots, n \quad \text{(A.8)}$$

$$X_{ijk} = 0{,}1 \qquad j = 1{,}2 \ldots, n_i, i = 1{,}2, \ldots, f, k = 1, 2, \ldots, n \quad \text{(A.9)}$$

$$Y_{iqk} = 0,1 \qquad k = 2,3,\ldots,n, i = 1,2\ldots,f, q = 1,2,\ldots,f \quad \text{(A.10)}$$

$$Y_{0q1} = 0,1 \qquad q = 1,2,\ldots,f \quad \text{(A.11)}$$

$$C_k, T_k \geq 0 \qquad k = 1,2,\ldots,n \quad \text{(A.12)}$$

"*The objective function is to minimize the total tardiness* (A.1). *Constraints* (A.2) *and* (A.3) *assure that one position can contain only one job and one job can be processed only once. Constraint* (A.4) *controls the setup time of the first position of the sequence, resulting in the completion time of the first position in* (A.6). *For any two consecutive jobs, Constraint* (A.5) *checks whether or not the preceding job and the following job are from the same family. If so, there is no setup time between them. Otherwise, setup time* s_{iq} *exists. Constraint* (A.7) *calculates the completion time from the* 2^{nd} *position to the last position of the sequence. Constraint* (A.8) *computes the tardiness value for each position. Note that* X_{ijk} *and* $Y_{i,q,k}$ *are binary integer variables*" (Chantaravarapan et al., 2003).

Apêndice B

Verificação dos pressupostos da Análise de Variância (ANOVA) para os experimentos do problema $1|ST_{sd,b}|\Sigma T_j$

Neste apêndice estão os demais testes e gráficos utilizados na condução dos diversos experimentos realizados. Constarão principalmente os testes de normalidade e igualdade de variância e a plotagem dos resíduos que são necessários de serem verificados quando na realização da ANOVA paramétrica.

Na verificação da igualdade de variância, que foi realizada pelo teste de *Levene*, que é mostrada nas tabelas com rótulo "*Variance Check*" nas legendas, o valor de interesse está na coluna "*P-Value*". Um valor para *P-Value* $>$ 0,05 indica que não há diferença estatisticamente significante entre os desvios padrões. O nível de significância nestes testes foi de 95%.

A verificação da normalidade realizada nos resíduos, foi feita pelos testes de *Shapiro-Wilk W* e *Chi Square*, e é mostrada nas tabelas com rótulo "*Tests for Normality*" nas legendas. O valor de interesse para este teste está na coluna "*P-Value*". Um valor para *P-Value* $>$ 0,05 indica que não se rejeita a hipótese de que os resíduos vem de uma distribuição normal. O nível de significância nestes testes foi de 95%.

Na calibração dos algoritmos ILS_BASIC, ILS_DP e ILS_DP+PR foram utilizadas 160, 256 e 256 configurações de parâmetros (conforme explicado na Seções 2.4.3.1, 2.4.3.2 e 2.4.3.3), na etapa 1 da calibração. Os resultados foram analisados

através da ANOVA. A tabela ANOVA e o resultado do *teste-F*, para cada um dos algoritmos, foram mostrados, respectivamente, nas Tabelas 2.3, 2.4 e 2.5. Nas Seções B.1, B.2 e B.3 constarão os demais testes, referentes a ANOVA, realizados na etapa 1 e na etapa 2 da calibração destes três algoritmos.

B.1 Análise de Variância para o experimento de calibração de parâmetros do algoritmo ILS_BASIC

Na etapa 1 da calibração do ILS_BASIC, a Tabela B.1 corresponde ao teste de igualdade de variância. A Tabela B.2 e o histograma de resíduos mostrado na Figura B.1 fornecem indícios suficientes para que se possa afirma que os resíduos vem de uma distribuição normal. A independência dos resíduos na ANOVA, significa que os erros ou resíduos não são correlacionados, e é verificada pela plotagem dos resíduos *versus* a ordem de observação. Para que os dados sejam considerados independentes não pode ser possível verificar qualquer tendência. A plotagem de resíduos dos resultados obtidos na etapa 1 da calibração do ILS_BASIC é mostrada na Figura B.2.

Tabela B.1. *Variance Check*. Etapa 1 da calibração do ILS_BASIC.

	Test	**P-Value**
Levene's	0,919849	**0,737421**

Tabela B.2. *Tests for Normality*. Etapa 1 da calibração do ILS_BASIC.

Test	**Statistic**	**P-Value**
Chi-Square	63,5	**0,13175**

Como explicado na Seção 2.4.3.1 foi necessário realizar mais uma etapa da calibração do ILS_BASIC. Os parâmetros neste segunda etapa também foram analisados através da ANOVA. O teste de igualdade de variância é mostrado na Tabela B.3. O teste de normalidade é mostrado na Tabela B.4 e na Figura B.3. A verificação da independência dos resíduos é mostrada na Figura B.4. A tabela ANOVA e o resultado do *teste-F*, para a etapa 2 da calibração do ILS_BASIC, é mostrada na Tabela B.5 e finalmente o gráfico de médias com intervalos HSD de *Tukey*, que mostra qual a melhor configuração para o ILS_BASIC, é exibida na Figura B.5.

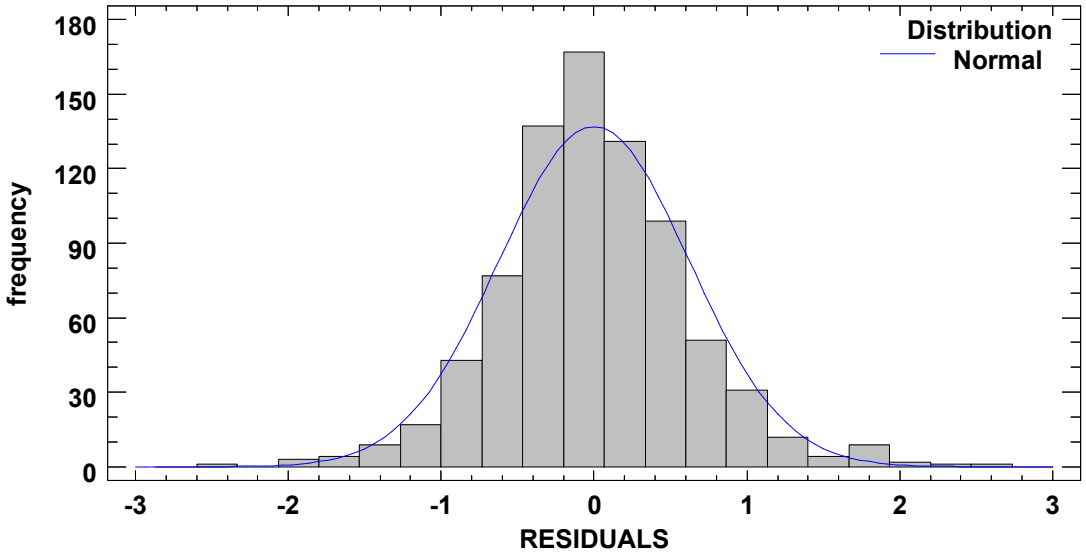


Figura B.1. Histograma e distribuição normal para os resíduos. Etapa 1 da calibração do ILS_BASIC.

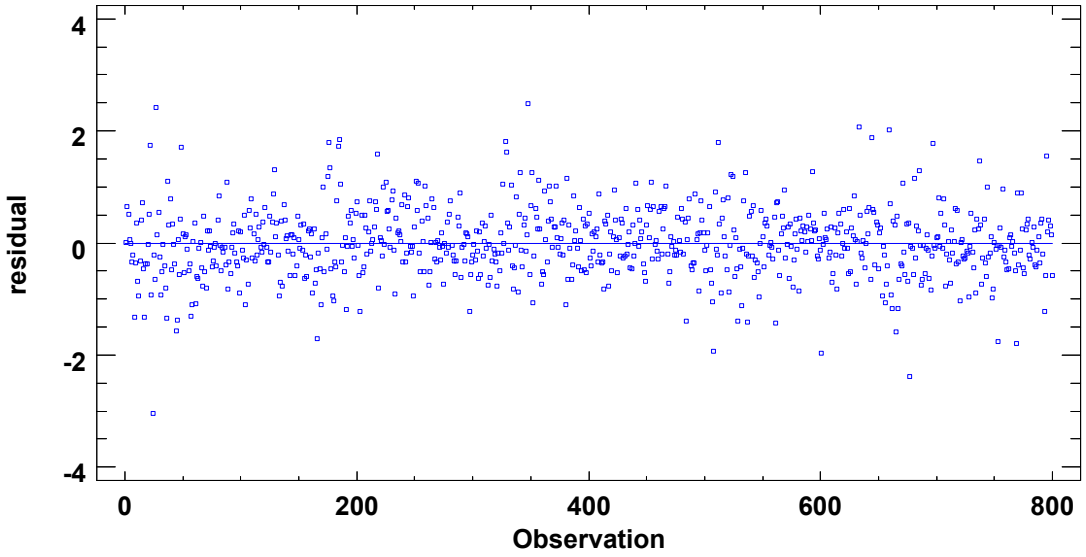


Figura B.2. Gráfico de resíduos *versus* observação. Etapa 1 da calibração do ILS_BASIC.

Tabela B.3. *Variance Check*. Etapa 2 da calibração do ILS_BASIC.

	Test	**P-Value**
Levene's	0,718083	**0,555568**

Tabela B.4. *Tests for Normality*. Etapa 2 da calibração do ILS_BASIC.

Test	**Statistic**	**P-Value**
Shapiro-Wilk W	0,967119	**0,688639**

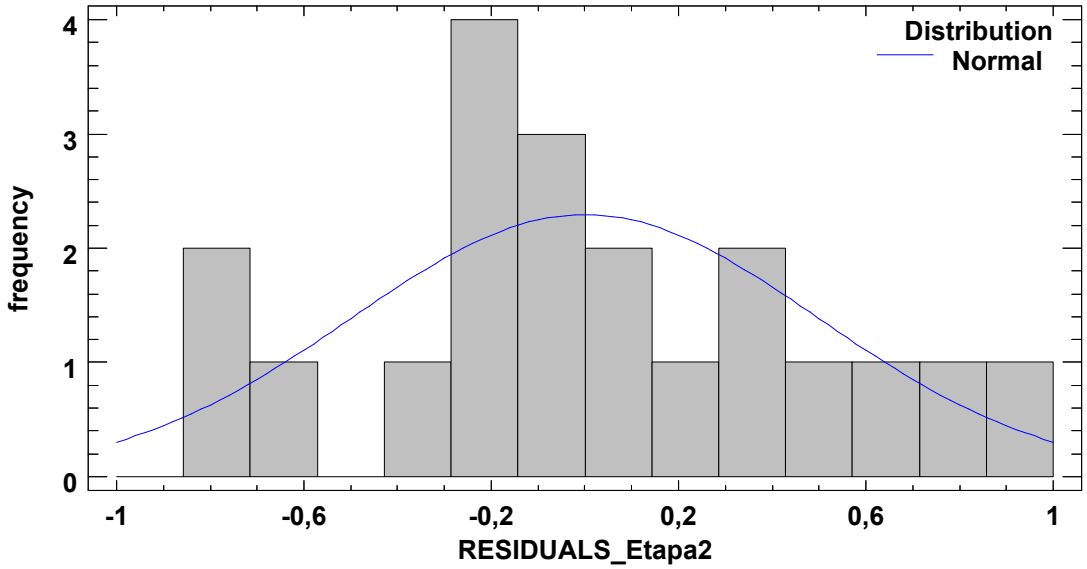


Figura B.3. Histograma e distribuição normal para os resíduos. Etapa 2 da calibração do ILS_BASIC.

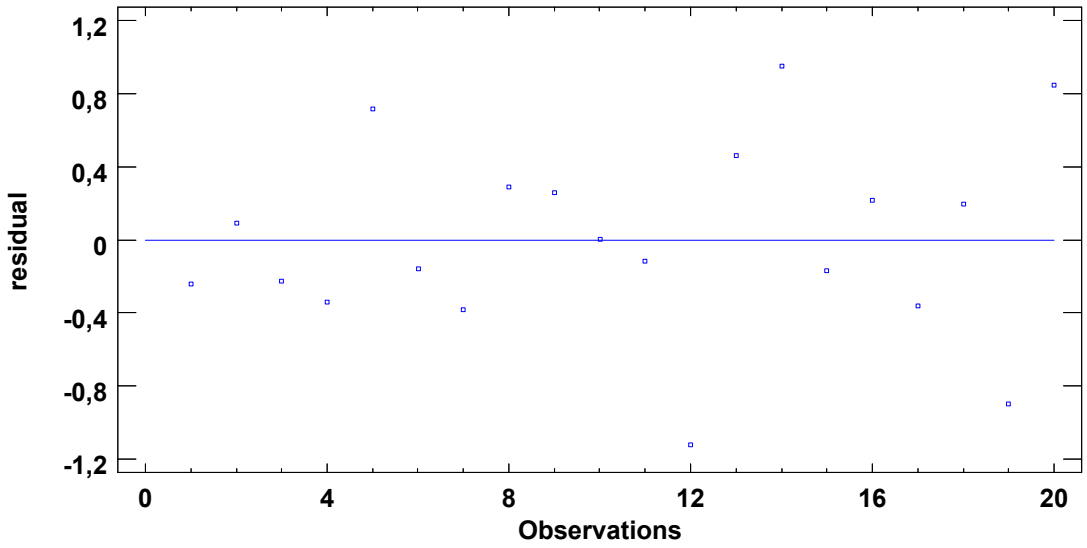


Figura B.4. Gráfico de resíduos *versus* observação. Etapa 2 da calibração do ILS_BASIC.

Tabela B.5. *ANOVA table*. Etapa 2 da calibração do ILS_BASIC.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	26,2728	3	8,75762	**26,90**	**0,0000**
Resíduos	5,20933	16	0,325583		
Total	31,4822	19			

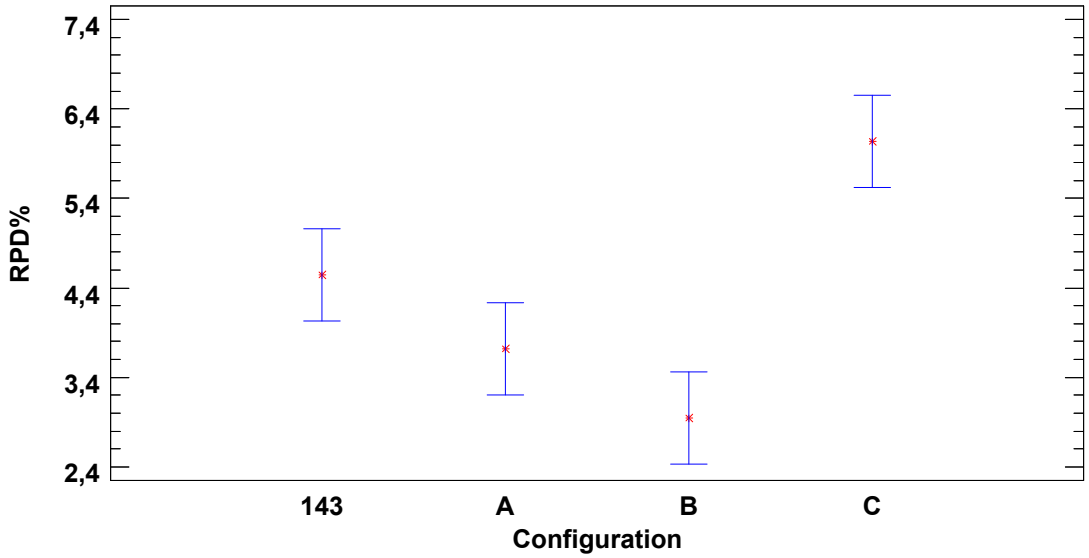


Figura B.5. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 4 configurações de ILS_BASIC testadas na etapa 2 da calibração. Na configuração A o parâmetro de entrada $d = \lceil \frac{n}{4} \rceil$; em B, $d = \lceil \frac{n}{3} \rceil$; em C, $d = \lceil \frac{n}{2} \rceil - 1$

B.2 Análise de Variância para o experimento de calibração de parâmetros do algoritmo ILS_DP

Na etapa 1 da calibração do ILS_DP, a Tabela B.6 corresponde ao teste de igualdade de variância. A Tabela B.7 e o histograma de resíduos mostrado na Figura B.6 fornecem indícios suficientes para que se possa afirma que os resíduos vem de uma distribuição normal. A independência dos resíduos na ANOVA é mostrada na Figura B.7.

Tabela B.6. *Variance Check.* Etapa 1 da calibração do ILS_DP.

	Test	**P-Value**
Levene's	0,98815	**0,539985**

Tabela B.7. *Tests for Normality.* Etapa 1 da calibração do ILS_DP.

Test	**Statistic**	**P-Value**
Shapiro-Wilk W	0,990926	**0,969291**

Como explicado na Seção 2.4.3.2 foi necessário realizar mais uma etapa da calibração do ILS_DP. Não foi verificado a homogeneidade de variância, conforme pode ser visto na Tabela B.8 e, por isso, foi realizado um teste de *Kruskal Wallis*, que é uma alternativa para o caso onde a ANOVA não pode ser aplicada.

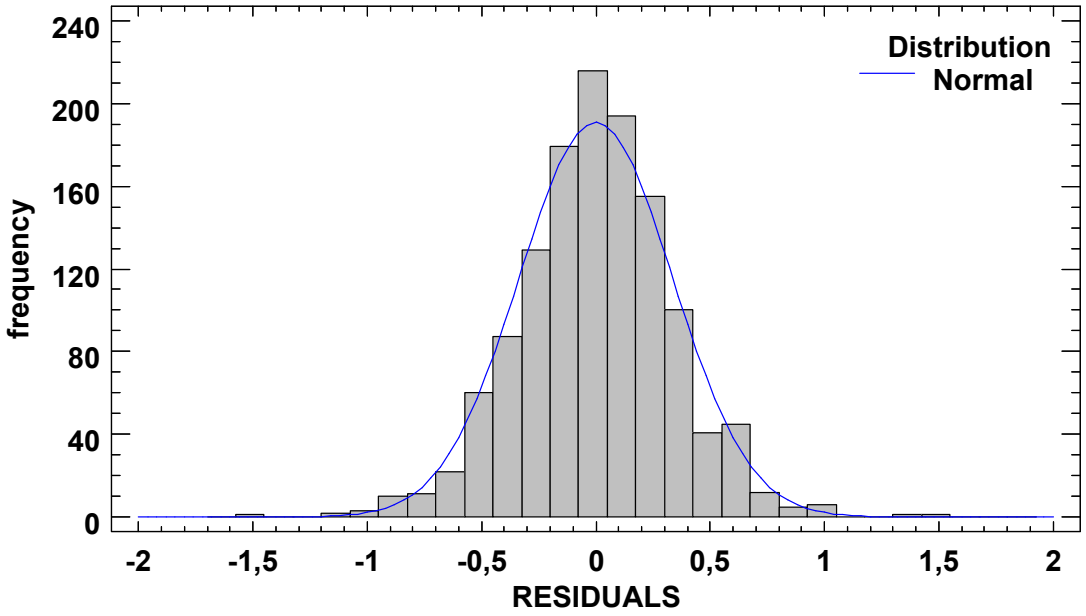


Figura B.6. Histograma e distribuição normal para os resíduos. Etapa 1 da calibração do ILS_DP.

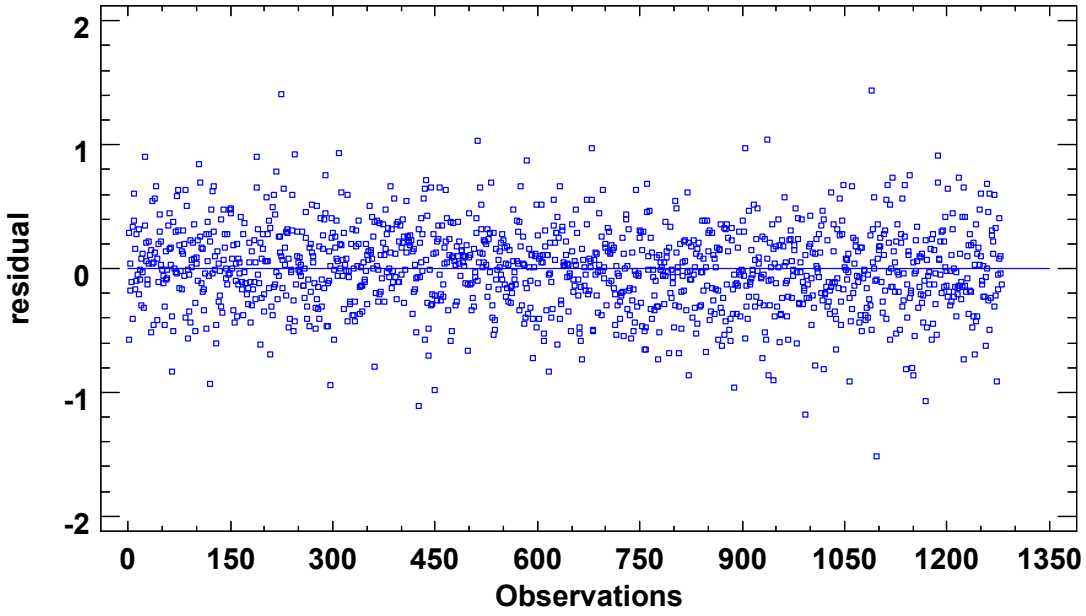


Figura B.7. Gráfico de resíduos *versus* observação. Etapa 1 da calibração do ILS_DP.

Tabela B.8. *Variance Check*. Etapa 2 da calibração do ILS_DP.

	Test	**P-Value**
Levene's	3,2397	**0,0499645**

O teste de *Kruskal Wallis* pode ser visto na Tabela B.9. Como o *valor-P*, que é o valor de interesse, é menor ou igual a 0,05 há uma diferença estatística significativa, no nível de significância considerado que é de 95%. Os contrastes entre os tratamentos é mostrado na Figura B.8 e indica que a configuração **B** é a melhor.

Tabela B.9. *Kruskal-Wallis Test.* Etapa 2 da calibração do ILS_DP. *Test statistic* = 10,7714. **P-Value = 0,0130278**

Parâmetros (Etapa 2)	Tamanho da amostra	*Average Rank*
142	5	16,6
A	5	11,2
B	5	4,4
C	5	9,8

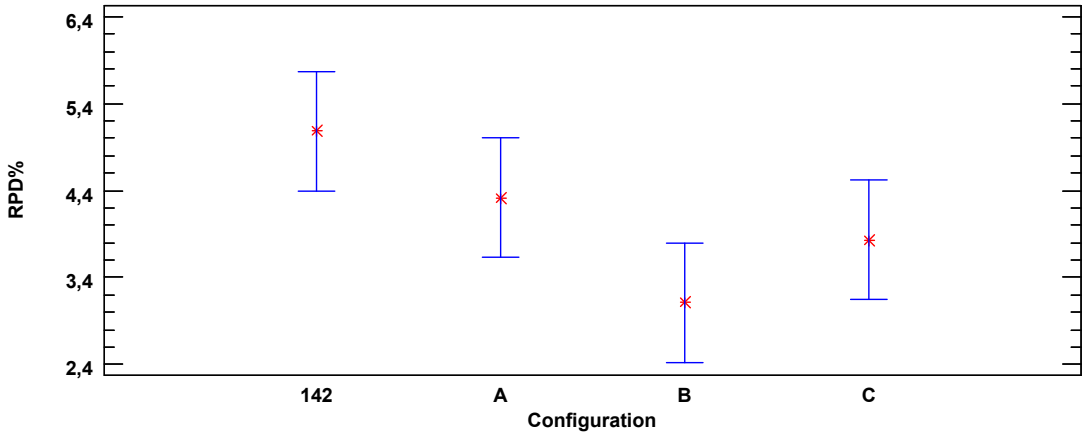


Figura B.8. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 4 configurações de ILS_DP testadas na etapa 2 da calibração. Na configuração A o parâmetro de entrada $d_{max} = \lceil \frac{n}{4} \rceil$; em B, $d_{max} = \lceil \frac{n}{3} \rceil$; em C $d_{max} = \lceil \frac{n}{2} \rceil - 1$

B.3 Análise de Variância para o experimento de calibração de parâmetros do algoritmo ILS_DP+PR

Na etapa 1 da calibração do ILS_DP+PR, a Tabela B.10 corresponde ao teste de igualdade de variância. A Tabela B.11 e o histograma de resíduos mostrado na Figura B.9 fornecem indícios suficientes para que se possa afirma que os resíduos vem de uma distribuição normal. A independência dos resíduos na ANOVA é mostrada na Figura B.10.

Tabela B.10. *Variance Check*. Etapa 1 da calibração do ILS_DP+PR.

	Test	P-Value
Levene's	0,825646	**0,969783**

Tabela B.11. *Tests for Normality*. Etapa 1 da calibração do ILS_DP+PR.

Test	Statistic	P-Value
Shapiro-Wilk W	0,987404	**0,534535**

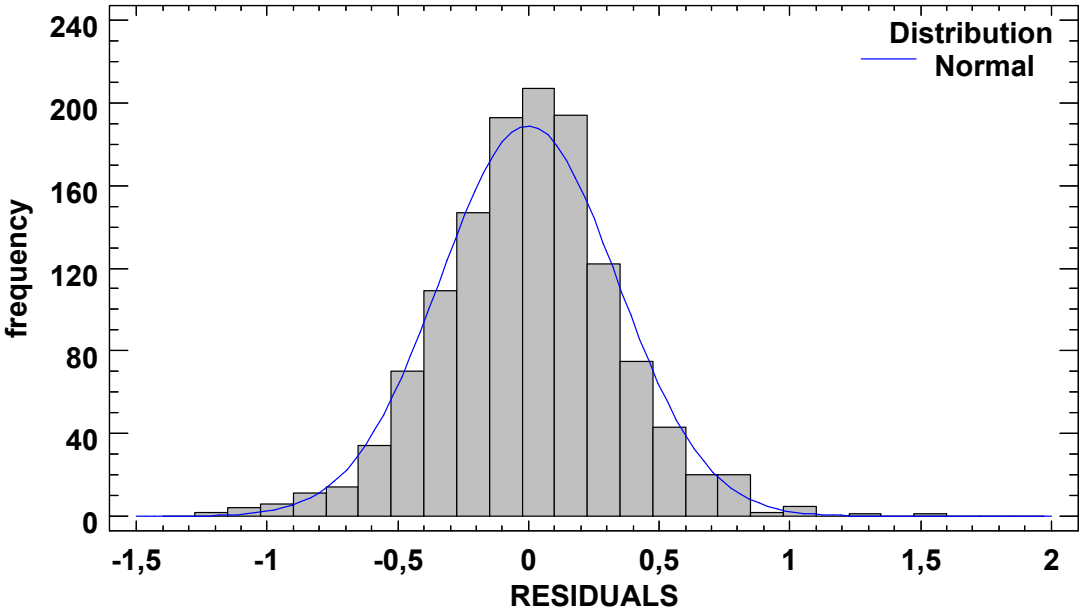


Figura B.9. Histograma e distribuição normal para os resíduos. Etapa 1 da calibração do ILS_DP+PR.

Como explicado na Seção 2.4.3.3 foi necessário realizar mais uma etapa da calibração do ILS_DP+PR. Os parâmetros neste segunda etapa também foram analisados através da ANOVA. O teste de igualdade de variância é mostrado na Tabela B.12. O teste de normalidade é mostrado na Tabela B.13 e na Figura B.11. A verificação da independência dos resíduos é mostrada na Figura B.12. A tabela ANOVA e o resultado do *teste-F*, para a etapa 2 da calibração do ILS_DP+PR, é mostrada na Tabela B.14 e finalmente o gráfico de médias com intervalos HSD de *Tukey*, que mostra qual a melhor configuração para o ILS_DP+PR, é exibida na Figura B.13.

Tabela B.12. *Variance Check*. Etapa 2 da calibração do ILS_DP+PR.

	Test	P-Value
Levene's	0,974379	**0,460644**

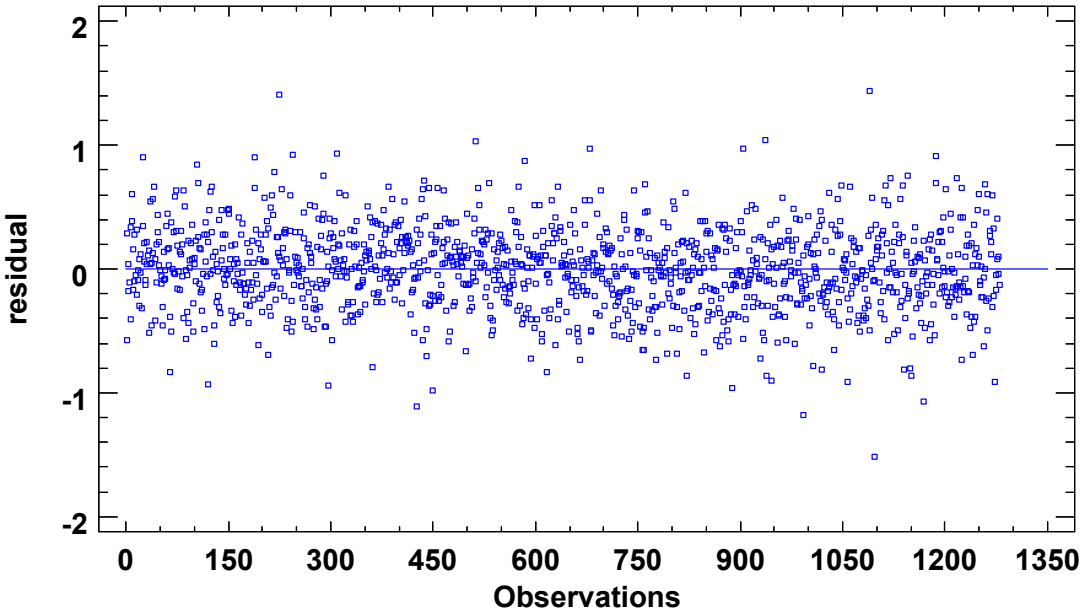


Figura B.10. Gráfico de resíduos *versus* observação. Etapa 1 da calibração do ILS_DP+PR.

Tabela B.13. *Tests for Normality*. Etapa 2 da calibração do ILS_DP+PR.

Test	Statistic	P-Value
Shapiro-Wilk W	0,978696	**0,779593**

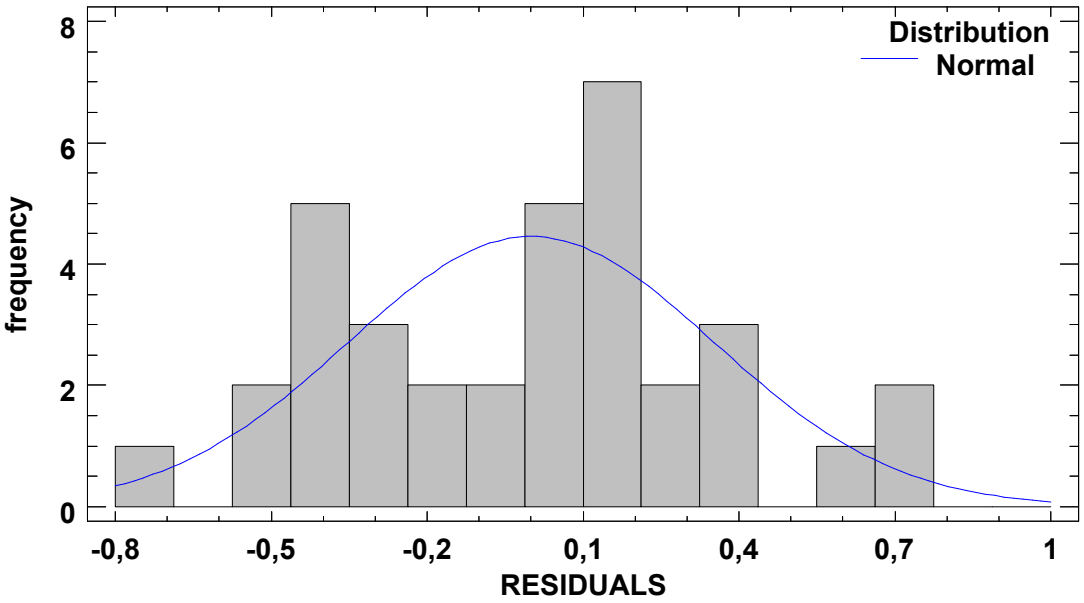


Figura B.11. Histograma e distribuição normal para os resíduos. Etapa 2 da calibração do ILS_DP+PR.

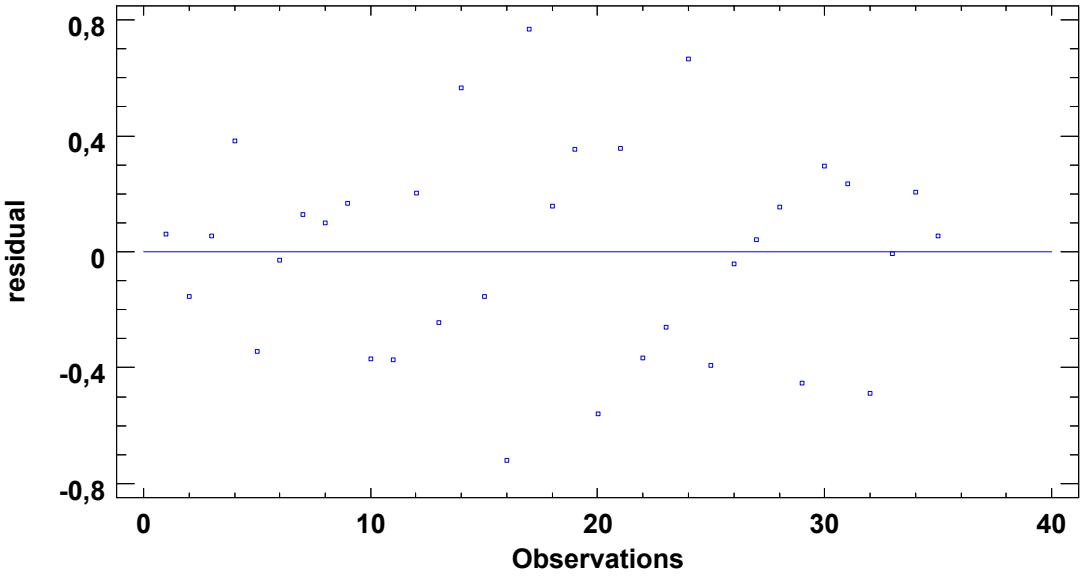


Figura B.12. Gráfico de resíduos *versus* observação. Etapa 2 da calibração do ILS_DP+PR.

Tabela B.14. *ANOVA table*. Etapa 2 da calibração do ILS_DP+PR.

Fonte da Variação	**Soma de Quadrados**	**Graus de Liberdade**	**Quadrados Médios**	**Valor de F**	**Valor-P**
Tratamentos	18,1348	6	3,02246	**20,03**	**0,0000**
Resíduos	4,22553	28	0,150912		
Total	22,3603	34			

B.4 Análise de Variância para o experimento de comparação dos algoritmos ILS_BASIC, ILS_DP e ILS_DP+PR

Nesta Seção estarão os testes realizados na comparação entre os algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR que foram apresentados na Seção 2.4.4. Os testes mostrados aqui corresponde a verificação dos pressupostos da ANOVA mostrada na Tabela 2.8. O teste de igualdade de variância é mostrado na Tabela B.15. O teste de normalidade é mostrado na Tabela B.16 e na Figura B.14. Finalmente, a verificação da independência dos resíduos é mostrada na Figura B.15.

Tabela B.15. *Variance Check*. Comparação entre os algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR

	Test	**P-Value**
Levene's	0,719552	**0,542262**

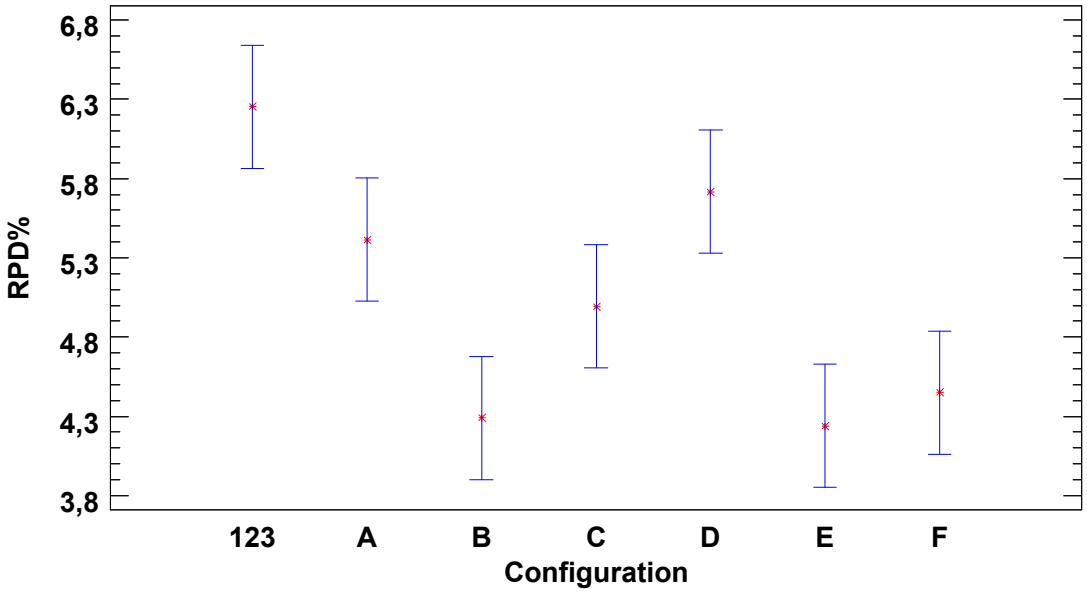


Figura B.13. Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as 4 configurações de ILS_DP+PR testadas na etapa 2 da calibração. Na configuração A o parâmetro de entrada $d_{max} = \lceil \frac{n}{4} \rceil$ e $sizeElite = 3$; em B, $d_{max} = \lceil \frac{n}{4} \rceil$ e $sizeElite = 5$; em C, $d_{max} = \lceil \frac{n}{3} \rceil$ e $sizeElite = 3$; em D, $d_{max} = \lceil \frac{n}{3} \rceil$ e $sizeElite = 5$; em E, $d_{max} = \lceil \frac{n}{2} \rceil - 1$ e $sizeElite = 3$; em F, $d_{max} = \lceil \frac{n}{2} \rceil - 1$ e $sizeElite = 5$

Tabela B.16. *Tests for Normality.* Comparação entre os algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR

Test	**Statistic**	**P-Value**
Shapiro-Wilk W	0,98126	**0,546023**

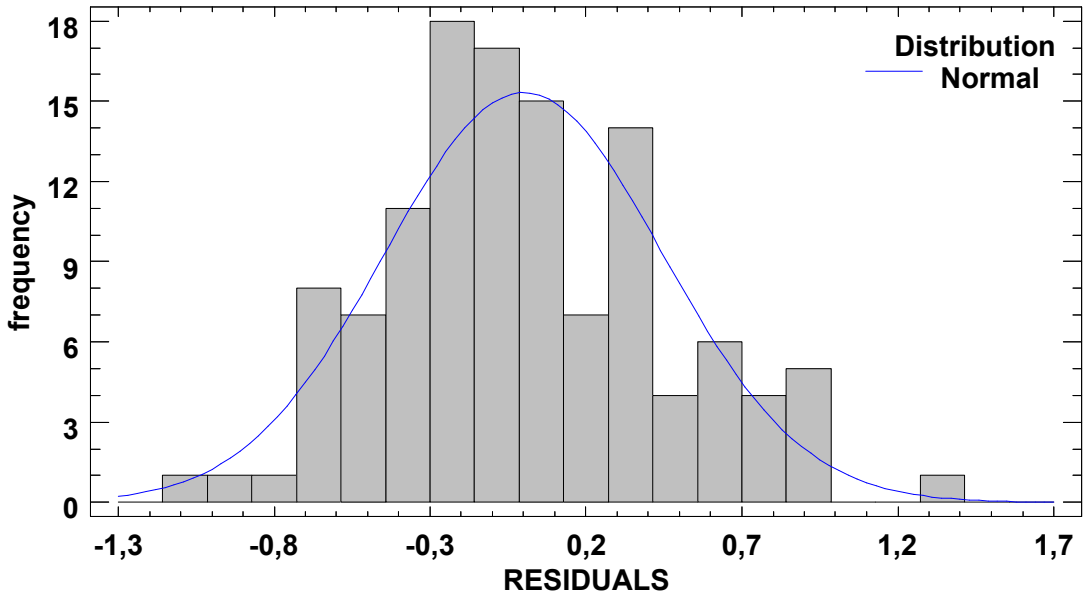


Figura B.14. Histograma e distribuição normal para os resíduos. Comparação entre os algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR

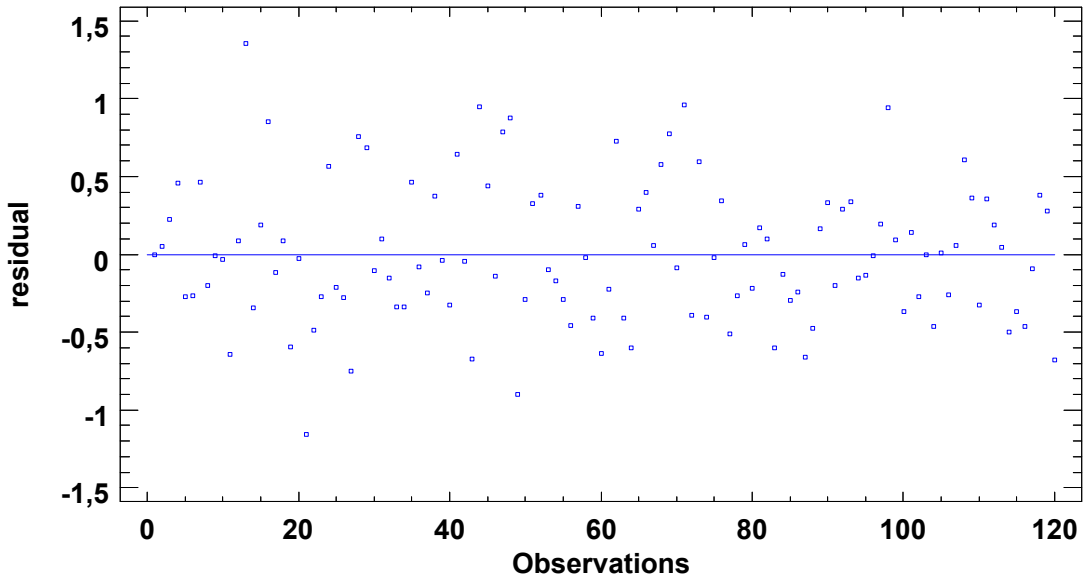


Figura B.15. Gráfico de resíduos *versus* observação. Comparação entre os algoritmos HGA, ILS_BASIC, ILS_DP e ILS_DP+PR

Apêndice C

Verificação dos pressupostos da Análise de Variância (ANOVA) para os experimentos do problema $1|r_{ij}|\Sigma w_{ij}F_{ij} + \Sigma\delta_i d_i$

C.1 Análise de Variância para o experimento de calibração de parâmetros do algoritmo ILS - instâncias de pequeno porte

Os testes e gráficos desta seção complementam os resultados mostrados na Seção 3.5.3, referentes à calibração do algoritmo ILS, que foi analisado através da ANOVA paramétrica, de modo que devem ser verificados os três pressupostos para que o *teste-F* da ANOVA mostrado na Tabela 3.5 tenha validade. O teste de igualdade de variância é mostrado na Tabela C.1. O teste de normalidade é mostrado na Tabela C.2 e na Figura C.1. A verificação da independência dos resíduos é mostrada na Figura C.2.

Tabela C.1. *Variance Check*. Calibração do ILS.

	Test	**P-Value**
Levene's	1,06065	**0,375836**

Tabela C.2. *Tests for Normality.* Calibração do ILS.

Test	Statistic	P-Value
Chi-Square	44,1333	**0,0932422**

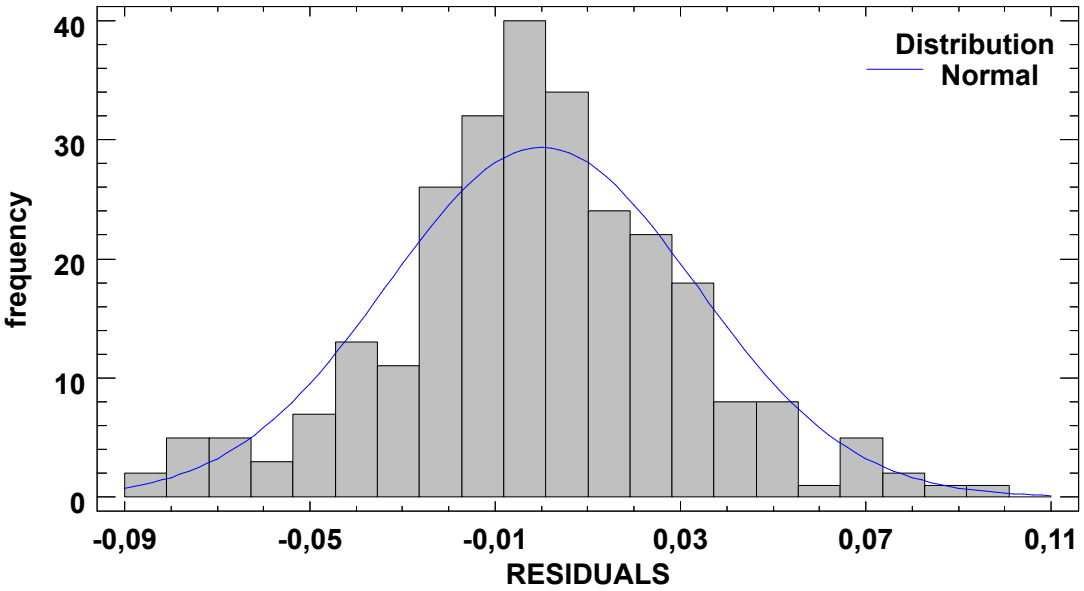


Figura C.1. Histograma e distribuição normal para os resíduos. Calibração do ILS.

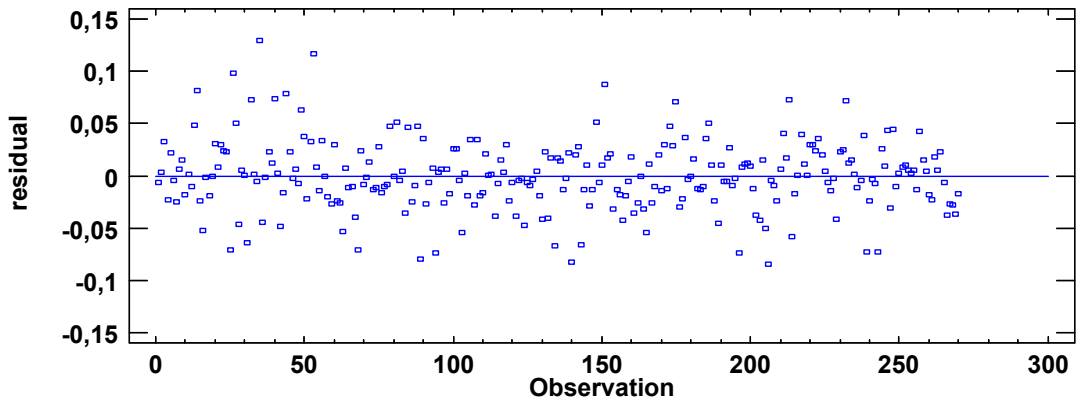


Figura C.2. Gráfico de resíduos *versus* observação. Calibração do ILS.

C.2 Análise de Variância para o experimento de calibração de parâmetros do algoritmo ILS - instâncias de grande porte

Os testes e gráficos desta seção complementam os resultados mostrados na Seção 3.5.6.3, referentes à calibração do algoritmo ILS para instâncias de grande porte,

que foi analisado através da ANOVA paramétrica, de modo que devem ser verificados os três pressupostos para que o *teste-F* da ANOVA mostrado na Tabela 3.18 tenha validade. O teste de igualdade de variância é mostrado na Tabela C.3. O teste de normalidade é mostrado na Tabela C.4 e na Figura C.3. A verificação da independência dos resíduos é mostrada na Figura C.4.

Tabela C.3. *Variance Check*. Calibração do ILS para instâncias de grande porte.

	Test	**P-Value**
Levene's	0,426869	**0,992711**

Tabela C.4. *Tests for Normality*. Calibração do ILS para instâncias de grande porte.

Test	**Statistic**	**P-Value**
Shapiro-Wilk W	0,976419	**0,255462**

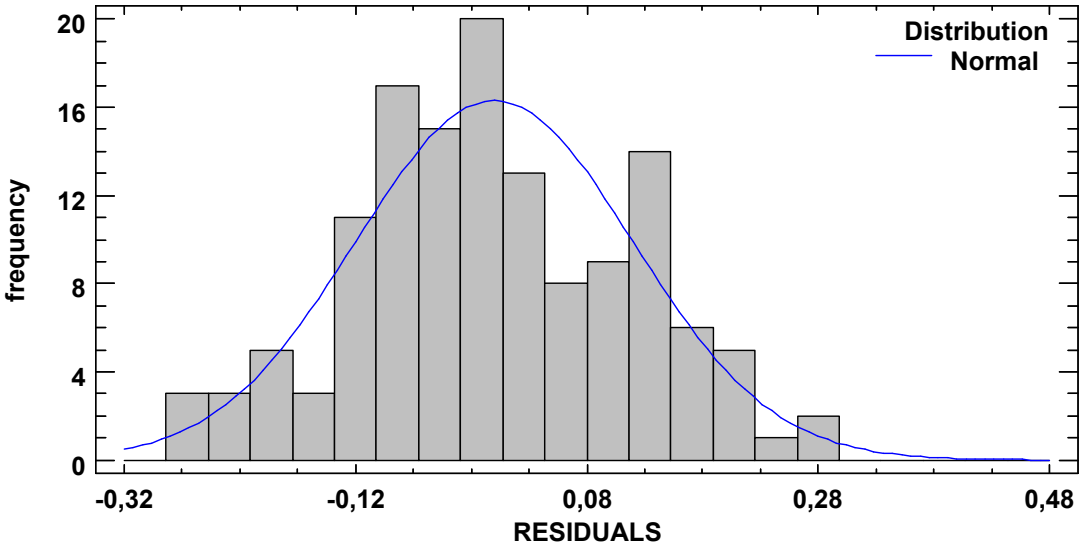


Figura C.3. Histograma e distribuição normal para os resíduos. Calibração do ILS para instâncias de grande porte.

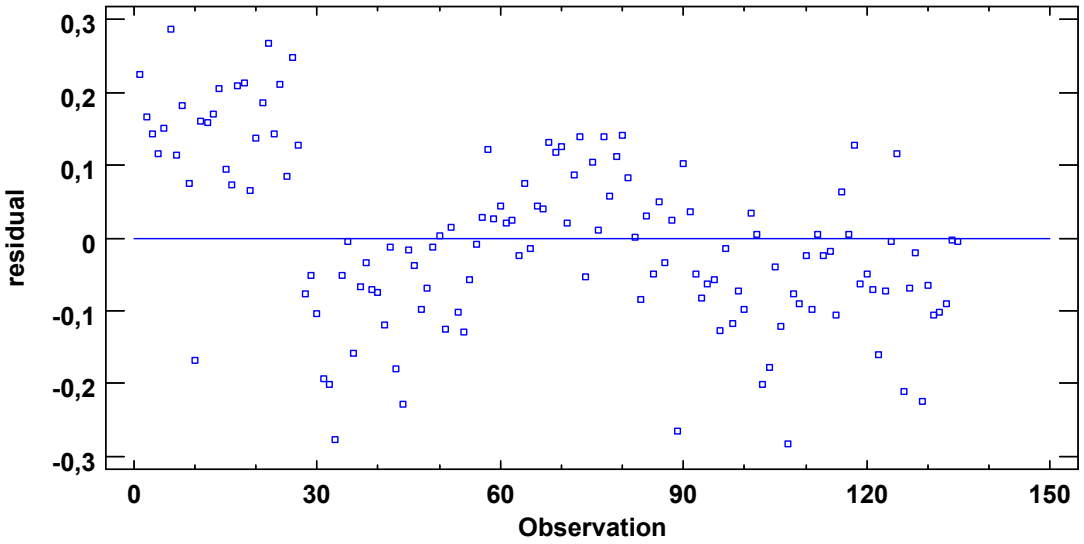


Figura C.4. Gráfico de resíduos *versus* observação. Calibração do ILS para instâncias de grande porte.

C.3 Análise de Variância para o experimento de calibração de parâmetros do algoritmo IG - instâncias de grande porte

Os testes e gráficos desta seção complementam os resultados mostrados na Seção 3.5.6.4, referentes à calibração do algoritmo IG para instâncias de grande porte, que foi analisado através da ANOVA paramétrica, de modo que devem ser verificados os três pressupostos para que o *teste-F* da ANOVA mostrado na Tabela 3.20 tenha validade. O teste de igualdade de variância é mostrado na Tabela C.5. O teste de normalidade é mostrado na Tabela C.6 e na Figura C.5. A verificação da independência dos resíduos é mostrada na Figura C.6.

Tabela C.5. *Variance Check.* Calibração do IG para instâncias de grande porte.

	Test	**P-Value**
Levene's	0,443927	**0,990293**

Tabela C.6. *Tests for Normality.* Calibração do IG para instâncias de grande porte.

Test	**Statistic**	**P-Value**
Shapiro-Wilk W	0,980314	**0,457823**

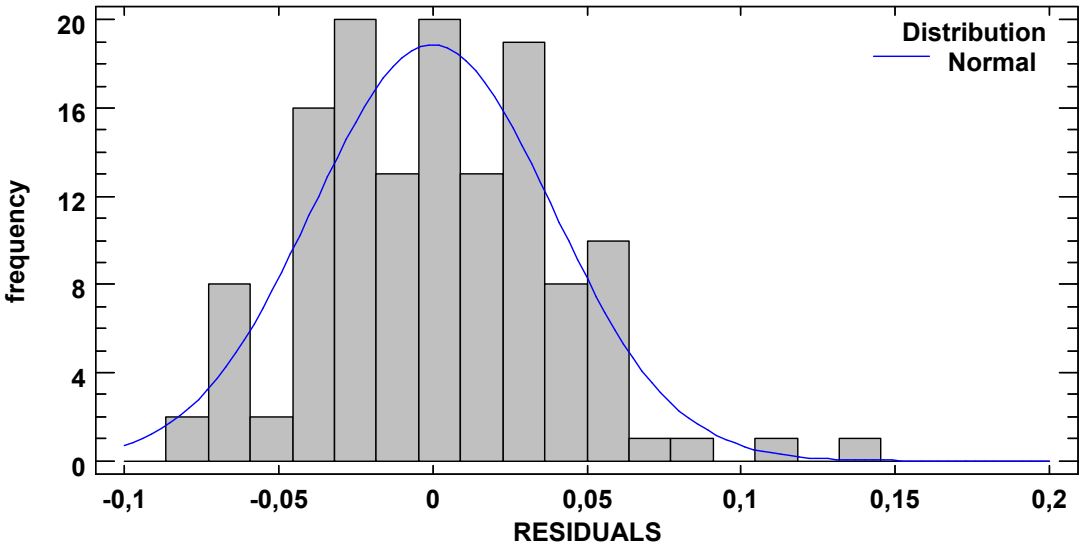


Figura C.5. Histograma e distribuição normal para os resíduos. Calibração do IG para instâncias de grande porte.

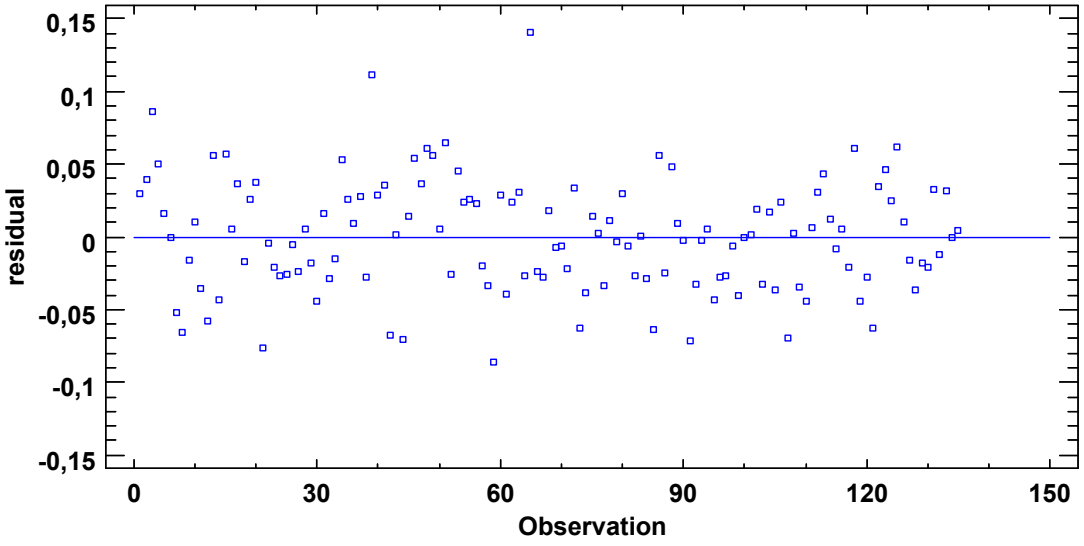


Figura C.6. Gráfico de resíduos *versus* observação. Calibração do IG para instâncias de grande porte.

C.4 Soluções ótimas encontradas através do MILP

Tabela C.7. Soluções ótimas encontradas através do MILP e a melhor solução e a solução média encontrada através do algoritmo ILS.

Instance	f_{OPT}	Best Solution ILS		Mean Solution ILS		Instance	f_{OPT}	Best Solution ILS		Mean Solution ILS	
		f_{ILS}	Error	f_{ILS}	Error			f_{ILS}	Error	f_{ILS}	Error
5-2-A-0	18912	18912	0,00%	18912,00	0,00%	7-3-B-6	5785	5785	0,00%	5785,00	0,00%
5-2-A-1	17218	17218	0,00%	17218,00	0,00%	7-3-B-7	11267	11267	0,00%	11267,00	0,00%
5-2-A-2	15096	15096	0,00%	15096,00	0,00%	7-3-B-8	14564	14564	0,00%	14564,00	0,00%
5-2-A-3	16411	16411	0,00%	16411,00	0,00%	7-3-B-9	4563	4563	0,00%	4563,00	0,00%
5-2-A-4	26272	26272	0,00%	26272,00	0,00%	8-4-A-0	37528	37528	0,00%	38133,27	1,61%
5-2-A-5	17009	17009	0,00%	17009,00	0,00%	8-4-A-1	43401	43401	0,00%	43401,00	0,00%
5-2-A-6	19114	19114	0,00%	19114,00	0,00%	8-4-A-2	33516	33516	0,00%	33829,17	0,93%
5-2-A-7	19245	19245	0,00%	19248,67	0,02%	8-4-A-3	34847	34847	0,00%	34847,00	0,00%
5-2-A-8	19572	19572	0,00%	19572,00	0,00%	8-4-A-4	40969	40969	0,00%	41395,70	1,04%
5-2-A-9	10390	10390	0,00%	10390,00	0,00%	8-4-A-5	23989	23989	0,00%	23989,00	0,00%
5-2-B-0	8058	8058	0,00%	8058,00	0,00%	8-4-A-6	34194	34194	0,00%	34299,23	0,31%
5-2-B-1	5522	5522	0,00%	5522,00	0,00%	8-4-A-7	42618	42618	0,00%	42996,90	0,89%
5-2-B-2	4904	4904	0,00%	4904,00	0,00%	8-4-A-8	22493	22493	0,00%	22518,30	0,11%
5-2-B-3	5729	5729	0,00%	5729,00	0,00%	8-4-A-9	42178	42178	0,00%	42178,00	0,00%
5-2-B-4	10505	10505	0,00%	10505,00	0,00%	8-4-B-0	10829	10829	0,00%	10862,73	0,31%
5-2-B-5	6875	6875	0,00%	6875,00	0,00%	8-4-B-1	10581	10581	0,00%	10581,00	0,00%
5-2-B-6	5751	5751	0,00%	5751,00	0,00%	8-4-B-2	9911	9911	0,00%	9911,00	0,00%
5-2-B-7	4120	4120	0,00%	4120,00	0,00%	8-4-B-3	12923	12923	0,00%	12923,00	0,00%
5-2-B-8	8382	8382	0,00%	8382,00	0,00%	8-4-B-4	16407	16407	0,00%	16407,00	0,00%
5-2-B-9	5272	5272	0,00%	5272,00	0,00%	8-4-B-5	7989	7989	0,00%	7989,00	0,00%
7-2-A-0	34075	34075	0,00%	34075,00	0,00%	8-4-B-6	14135	14135	0,00%	14273,27	0,98%
7-2-A-1	29768	29887	0,40%	29887,00	0,40%	8-4-B-7	15134	15134	0,00%	15134,67	0,00%
7-2-A-2	22428	22428	0,00%	22428,00	0,00%	8-4-B-8	6320	6343	0,36%	6379,60	0,94%
7-2-A-3	17230	17230	0,00%	17230,00	0,00%	8-4-B-9	12689	12689	0,00%	12689,00	0,00%
7-2-A-4	20783	20783	0,00%	20834,13	0,25%	10-2-A-2	34647	34647	0,00%	34659,10	0,03%
7-2-A-5	17101	17101	0,00%	17101,00	0,00%	10-2-A-3	38519	38681	0,42%	39034,40	1,34%
7-2-A-6	25113	25266	0,61%	25266,00	0,61%	10-2-A-6	24423	24423	0,00%	24423,00	0,00%
7-2-A-7	19926	19926	0,00%	19926,00	0,00%	10-2-A-9	13380	13483	0,77%	13483,00	0,77%
7-2-A-8	19348	19348	0,00%	19348,00	0,00%	10-2-B-9	7936	7936	0,00%	7936,00	0,00%
7-2-A-9	26077	26077	0,00%	26126,40	0,19%	10-3-A-2	40119	40295	0,44%	40295,00	0,44%
7-2-B-0	14271	14271	0,00%	14271,00	0,00%	10-3-A-5	45131	45131	0,00%	45131,00	0,00%
7-2-B-1	9798	9798	0,00%	9798,00	0,00%	10-3-A-6	31501	31780	0,89%	31780,00	0,89%
7-2-B-2	8880	8880	0,00%	8880,00	0,00%	10-3-A-9	19949	19971	0,11%	19971,00	0,11%
7-2-B-3	6142	6142	0,00%	6155,60	0,22%	10-3-B-9	8404	8404	0,00%	8413,60	0,11%
7-2-B-4	11570	11570	0,00%	11570,00	0,00%	10-4-A-0	36774	36774	0,00%	36774,00	0,00%
7-2-B-5	10173	10173	0,00%	10173,00	0,00%	10-4-A-1	39400	39546	0,37%	39572,13	0,44%
7-2-B-6	8092	8092	0,00%	8092,00	0,00%	10-4-A-2	34841	34841	0,00%	34841,00	0,00%
7-2-B-7	5555	5555	0,00%	5555,00	0,00%	10-4-A-4	49052	49052	0,00%	49052,00	0,00%
7-2-B-8	14326	14326	0,00%	14326,00	0,00%	10-4-A-5	35743	35743	0,00%	35743,00	0,00%
7-2-B-9	4186	4186	0,00%	4186,00	0,00%	10-4-A-8	19882	19882	0,00%	20353,43	2,37%
7-3-A-0	28908	28908	0,00%	28908,00	0,00%	10-4-B-1	12642	12642	0,00%	12642,00	0,00%
7-3-A-1	29369	29369	0,00%	29369,00	0,00%	10-4-B-8	8142	8142	0,00%	8156,40	0,18%
7-3-A-2	17701	17701	0,00%	17701,00	0,00%	10-5-A-0	48598	48710	0,23%	48794,33	0,40%
7-3-A-3	28225	28397	0,61%	28397,00	0,61%	10-5-A-1	38387	38387	0,00%	38387,00	0,00%
7-3-A-4	21669	21669	0,00%	21669,00	0,00%	10-5-A-2	29642	30047	1,37%	30161,33	1,75%
7-3-A-5	29297	29297	0,00%	29297,00	0,00%	10-5-A-3	52784	52784	0,00%	52992,60	0,40%
7-3-A-6	22296	22296	0,00%	22296,00	0,00%	10-5-A-4	40783	40783	0,00%	40871,30	0,22%
7-3-A-7	22484	22484	0,00%	22484,00	0,00%	10-5-A-5	41662	41662	0,00%	41840,90	0,43%
7-3-A-8	25870	25870	0,00%	25870,00	0,00%	10-5-A-6	51696	51696	0,00%	51768,33	0,14%
7-3-A-9	25523	25523	0,00%	25523,00	0,00%	10-5-A-7	20935	21034	0,47%	21034,00	0,47%
7-3-B-0	10056	10056	0,00%	10056,00	0,00%	10-5-A-8	39410	39410	0,00%	39600,67	0,48%
7-3-B-1	8337	8337	0,00%	8337,00	0,00%	10-5-A-9	37434	37434	0,00%	37511,07	0,21%
7-3-B-2	6453	6453	0,00%	6453,00	0,00%	10-5-B-0	13528	13528	0,00%	13528,00	0,00%
7-3-B-3	11851	11851	0,00%	11851,00	0,00%	10-5-B-4	9488	9488	0,00%	9492,33	0,05%
7-3-B-4	10443	10443	0,00%	10443,00	0,00%	10-5-B-7	7377	7377	0,00%	7377,00	0,00%
7-3-B-5	8956	8956	0,00%	8956,00	0,00%	15-5-A-7	51393	51393	0,00%	52007,23	1,20%

Apêndice D

Reimplementação do algoritmo *Branch and Bound*

Neste apêndice estão descritos possíveis pontos de divergência do trabalho realizado por Mazdeha et al. (2012), uma vez que não se conseguiu provar a otimalidade das soluções obtidas através do algoritmo *Branch and Bound* (B&B) por meio de sua reimplementação, que seguiu as diretrizes do artigo original. A reimplementação foi necessária porque não foi possível obter a implementação diretamente por solicitação ao autor do trabalho supracitado.

Primeiramente, as proposições aplicadas para calcular o LBF, no exemplo numérico, parecem inconsistentes com as proposições apresentadas no próprio artigo, principalmente as proposições 2 e 3. LBF é um limitante inferior associado ao cálculo do fluxo ponderado total utilizado no algoritmo B&B (mais detalhes consultar o artigo original).

A árvore da Figura D.1 mostra as soluções exploradas no exemplo numérico. Foi utilizada uma notação semelhante a do trabalho original e os pontos de divergência são explicitamente marcados com tachado seguido do valor considerado correto e que foi utilizado neste trabalho. Em cada nodo são apresentados os valores do limitante inferior **LB** seguindo da proposição **P** utilizada no cálculo do LBF. Nesta figura P1, significa utilização da proposição 1, P2 e A1 significa utilização da proposição 2 e algoritmo 1 e P3 significa utilização da proposição 3. P1, P2, P3 e A1 referem-se a proposições e algoritmos do trabalho original de Mazdeha et al. (2012). No trabalho original não foi explicitado quais proposições foram utilizadas no cálculo do LBF para os nodos S0, S1, S2, S4, S6 e S8.

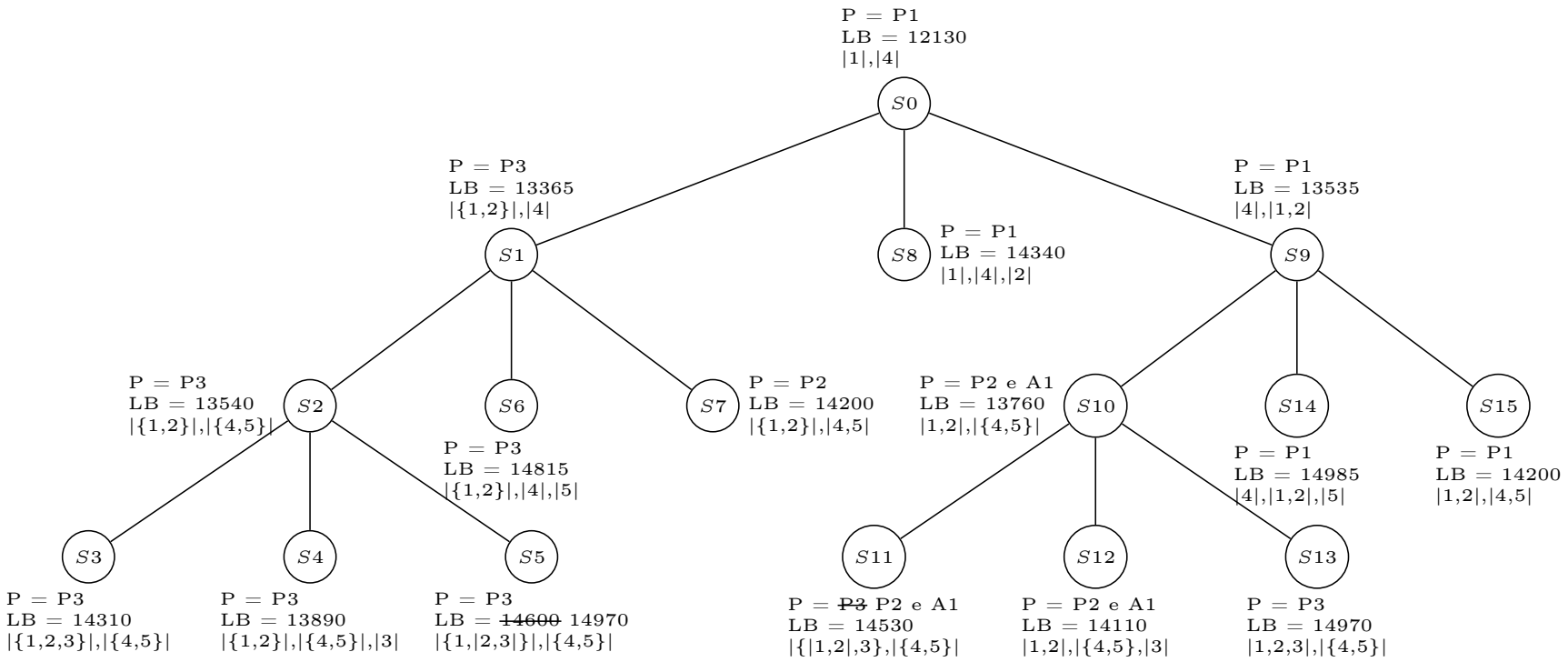


Figura D.1. Árvore de busca do algoritmo *Branch and Bound*. Observe que "|" neste contexto não significa cardinalidade de conjuntos mas sim uma notação utilizada no trabalho Mazdeha et al. (2012) para diferenciar lotes contínuos e lotes descontínuos.

Outro problema encontrado, que dificultou a implementação do algoritmo B&B, está relacionado a proposição 5 e corolário 2. O autor menciona na proposição 5: "*na solução ótima, qualquer lote b com mais do que uma tarefa, destinada ao cliente 'j' irá ter a propriedade* $(W_b - w_l) \times p_l \leq d_j$*, onde* W_b *é definido como a soma dos pesos das tarefas no lote b e* w_l *e* p_l *são o peso e o tempo de processamento da última tarefa no lote b, respectivamente*". E no corolário 2, o autor afirma: "*na solução ótima, qualquer tarefa [k] que tenha a propriedade* $p_k > d_j$ *irá formar um lote com uma única tarefa*" (Mazdeha et al., 2012, nossa tradução).

O problema ocorre porque a propriedade mencionada na proposição 5, $(W_b - w_l) \times p_l \leq d_j$, é válida somente para um caso particular onde as duas últimas tarefas do lote b estão dispostas continuamente. Como demonstrado a seguir, para um caso geral a propriedade deveria ser $(W_b - w_l) \times (E_l - E_k) \leq d_j$, onde E_l é o tempo de conclusão da última tarefa do lote b e E_k é o tempo de conclusão da penúltima tarefa do lote b.

A demostração seguirá a mesma ideia do artigo original. Considere um lote que não tenha a propriedade $(W_b - w_l) \times (E_l - E_k) \leq d_j$. Removendo a última tarefa J_l e entregando ela num lote com uma única tarefa a função objetivo como um todo irá diminuir pelo menos por $(W_b - w_l) \times (E_l - E_k)$. A Figura D.2 ilustra a situação onde a tarefa j_l é entregue junto com as demais tarefas do lote b. Nesta situação a função objetivo F_O pode ser descrita como: $F_O = F' + W_b E_l + d_j$, onde F' é a parcela (considerando fluxo ponderado e custos de entrega) somada à função objetivo associada a todas as outras tarefas que não estão no lote b. Agora considere outra situação, ilustrada na Figura D.3, onde a tarefa j_l, que era a última tarefa do lote b, será entregue separadamente em um lote 'b_1' e as demais tarefas que pertenciam ao lote b agora compõem o lote 'b_2'. Neste caso o novo valor da função objetivo F'_O será dado por: $F'_O = F' + (W_b - w_l)E_k + E_l w_l + 2d_j$.

$F_O \leq F'_O$
$\Rightarrow F' + W_b E_l + d_j \leq F' + (W_b - w_l)E_k + E_l w_l + 2d_j$
$\Rightarrow W_b E_l - (W_b - w_l)E_k - E_l w_l \leq d_j$
$\Rightarrow W_b E_l - W_b E_k + w_l E_k - E_l w_l \leq d_j$
$\Rightarrow W_b(E_l - E_k) - w_l(E_l - E_k) \leq d_j$
$\Rightarrow (W_b - w_l)(E_l - E_k) \leq d_j \quad \square$

O caso particular que é abordado no trabalho de Mazdeha et al. (2012), ocorre quando as tarefas j_k e j_l são contínuas. Neste caso $E_l - E_k = p_l$, de modo que $(W_b - w_l)(E_l - E_k) - d_j \leq 0$ implique $(W_b - w_l)p_l \leq d_j$.

Segundo o corolário 2, qualquer tarefa j_k, pertencente ao cliente j, e que possua a propriedade $p_k > d_j$ será entregue em um lote com uma única tarefa

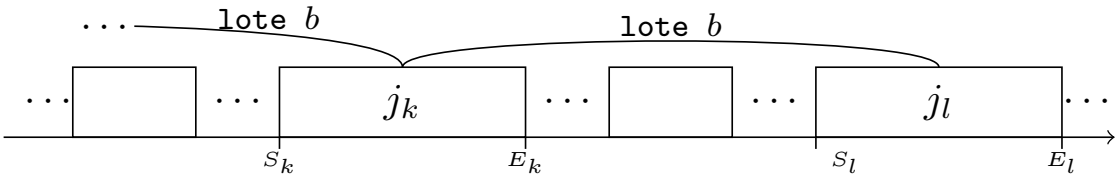


Figura D.2. Situação onde a tarefa j_l é entregue junto com as tarefas do lote b

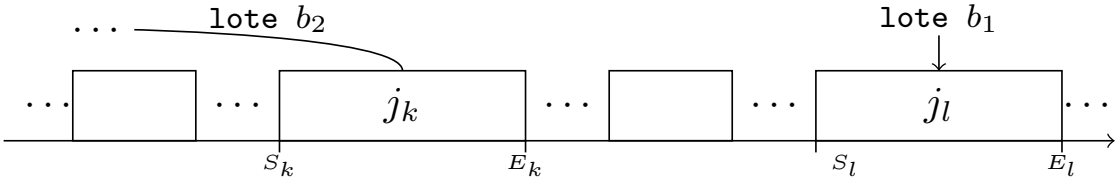


Figura D.3. Situação em que a tarefa j_l é entregue em um lote separado onde ela é a única tarefa.

(ela própria). E na explicação do algoritmo *Branch and Bound* é dito que "*todas as tarefas que formam um lote com uma única tarefa em virtude do corolário 2, são identificadas e separadas das outras tarefas e à sua variável correspondente é atribuído o valor zero*". O algoritmo considera que um determinado nodo (ou solução parcial) é ramificado em outros três: "*o primeiro [nodo] indica o início de um novo lote (0), o segundo [nodo] indica que uma nova tarefa é adicionada a* ***um lote aberto*** *continuamente (1) e o terceiro [nodo] indica adicionar uma tarefa a* ***um lote aberto*** *descontinuamente (2)*". Porém, não fica claro em qual lote a tarefa será adicionada e nem o que significa o termo "lote aberto" (*open batch*). Tudo leva a crer, baseando-se principalmente no exemplo numérico apresentado no trabalho original, que o termo "lote aberto" é um lote cuja variável não foi atribuído o valor zero, ou seja, um lote que em virtude do corolário 2 não foi atribuído o valor zero. Na implementação utilizada no presente trabalho, a tarefa é adicionada sempre ao último "lote aberto" e cujo cliente seja o mesmo da tarefa a ser adicionada ao nodo da árvore para compor as novas soluções.

Ainda com relação ao corolário 2, é possível demonstrar que, mesmo que uma tarefa j_k possua a propriedade $p_k > d_j$, pode ser melhor entregar a tarefa j_k, para o cliente j, em um lote com mais de uma tarefa, conforme mostrado a seguir. Considere uma solução qualquer, onde existe uma determinada tarefa j_k que possua a propriedade $p_k > d_j$ e que seja a penúltima tarefa do lote b. j_l é a última tarefa do lote b e é processada imediatamente depois da tarefa j_k, conforme ilustrado

na Figura D.4. Neste caso, o valor da função objetivo F_1 pode ser descrito como $F_1 = F' + W_b(E_k + p_l) + d_j$, onde F' é a parcela (considerando fluxo ponderado e custos de entrega) somada à função objetivo associada a todas as outras tarefas que não estão no lote b. Se a tarefa j_k for retirada do lote b e for entregue em um lote separado o novo valor da função objetivo F_2 será dado por: $F_2 = F' + w_k E_k + (W_b - w_k)(E_k + p_l) + 2d_j$. Pode existir um caso em que é melhor colocar j_k no lote b que é quando $w_k p_l \leq d_j$:

$F_1 \leq F_2$
$\Rightarrow F' + W_b(E_k + p_l) + d_j \leq F' + w_k E_k + (W_b - w_k)(E_k + p_l) + 2d_j$
$\Rightarrow W_b(E_k + p_l) \leq w_k E_k + W_b(E_k + p_l) - w_k(E_k + p_l) + d_j$
$\Rightarrow w_k p_l \leq d_j$

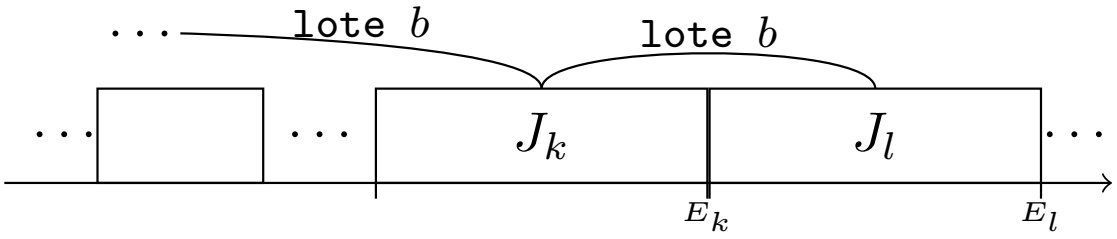


Figura D.4. Solução onde as tarefas j_k e j_l pertencem ao mesmo lote b e j_l é a última tarefa processada seguindo imediatamente o processamento da tarefa j_k

Outro problema enfrentado foi com a proposição 3, que aparentemente não leva ao valor ótimo do fluxo ponderado total. Ao realizar o experimento ocorreu de em uma das instâncias o valor encontrado pelo algoritmo B&B ser superior ao obtido através do MILP proposto neste trabalho. O B&B encontrou o valor 19971 enquanto que o MILP provou que o valor ótimo seria de 19949. Os dados de entrada deste problema teste são mostrados na Tabela D.1.

Tabela D.1. Dados de entrada para a instância '10-3-A-9' que considera 10 tarefas e 3 clientes. O custo de entrega para os clientes 0, 1 e 2 são $d_0 = 2460$, $d_1 = 2993$ e $d_2 = 6529$, respectivamente

Tarefa j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
p_j	1	11	10	25	19	48	36	23	44	51
r_j	5	10	11	20	44	52	58	58	75	85
w_j	6	8	3	6	3	7	4	2	3	2
c_j	1	1	0	1	0	1	2	2	2	1
p_j/w_j	0,17	1,38	3,33	4,17	6,33	6,86	9	11,5	14,67	25,5

A solução provada ser ótima pelo MILP possui a seguinte conformação de lotes: |{2,4}|, |{0,3,1,5,9}|, |{6,7,8}| e é demostrada na Figura D.5. Ao analisar a solução obtida através do B&B a formação de lotes é igual àquela encontrada através do MILP, porém o sequenciamento de tarefas, que de acordo com a proposição 3 deveria ser determinado pela regra WSERBT (*Weighted Shortest Effective Remaining Batch Time*), foi diferente causando a divergência entre os valores. A solução obtida através do algoritmo B&B é exibida na Figura D.6

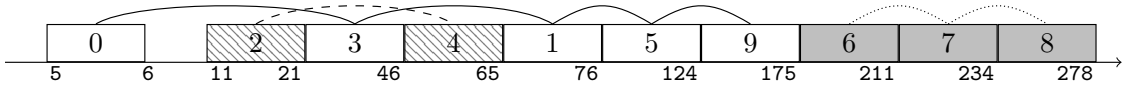


Figura D.5. Solução provada ser ótima pelo MILP para o problema teste '10-3-A-9' e que possui o custo de 19949.

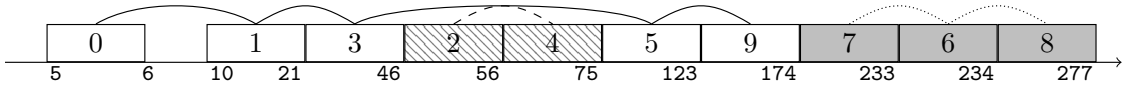


Figura D.6. Solução obtida através do *Branch and Bound* para o problema teste '10-3-A-9' e que possui o custo de 19971.

Por fim, não foi deixado explícito em nenhum momento no trabalho de Mazdeha et al. (2012) como uma solução foi representada na implementação. Então, por questão de reprodutibilidade de experimento será descrita, em alto nível, a representação utilizada para a implementação do algoritmo B&B que fora realizada neste trabalho.

Cada **nodo** na árvore a ser explorada no B&B contém uma **solução**, que pode ser uma solução parcial ou uma solução completa que ocorre nos nodos folhas. Na implementação realizada não existe explicitamente o conceito de tarefa, sendo que a solução é composta por um **conjunto de lotes**. Um lote por sua vez, recursivamente, também é formado por um conjunto de lotes. Esta representação se aproxima da notação utilizada por Mazdeha et al. (2012) para descrever uma formação de lotes. Por exemplo, a formação de lotes |{1,|2,3|}| na notação de Mazdeha et al. (2012) é representada na implementação como um lote que contém dois outros lotes: o lote formado exclusivamente pela tarefa 1 e pelo lote que contém as tarefas 2 e 3. O lote |2,3| por sua vez é composto por dois lotes: o lote formado exclusivamente pela tarefa 2 e o lote formado exclusivamente pela tarefa 3. É utilizado também o conceito de "lote pai" e "lote filho", sendo que cada lote armazena uma

referência ao seu lote "pai" e aos seus "filhos". Então, por exemplo, o lote |{1,|2,3|}| possui somente filhos sendo ao pai do lote |{1,|2,3|}| atribuído o valor NULO. O lote |{1,|2,3|}| possui dois lotes filhos: |1| e |2,3|. O lote |2,3| também possui dois lotes filhos: |2| e |3|. Os lotes |1|, |2| e |3| por sua vez não possuem filhos sendo atribuído o valor NULO à variável correspondente.