ANA AMÉLIA DE SOUZA PEREIRA

METAHEURÍSTICAS PARA O PROBLEMA DE *FLOWSHOP* FLEXÍVEL COM PENALIDADES DE ADIANTAMENTO E ATRASO

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
2011

Ficha catalográfica preparada pela Seção de Catalogação e Classificação da Biblioteca Central da UFV

T
P436m
2011

Pereira, Ana Amélia de Souza, 1977-
Metaheurísticas para o problema de *Flowshop* flexível com penalidades de adiantamento e atraso / Ana Amélia de Souza Pereira. – Viçosa, MG, 2011.
xii, 57f. : il. (algumas col.) ; 29cm.

Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 54-57

1. Pesquisa operacional. 2. Otimização combinatória .
3. Planejamento da produção. 4. Programação heurística.
I. Universidade Federal de Viçosa. II. Título.

CDD 22. ed. 003

ANA AMÉLIA DE SOUZA PEREIRA

METAHEURÍSTICAS PARA O PROBLEMA DE *FLOWSHOP* FLEXÍVEL COM PENALIDADES DE ADIANTAMENTO E ATRASO

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: 02 de agosto de 2011

Prof.: Mauro Nacif Rocha

Prof.: Heleno do Nascimento Santos

Prof.: José Elias Claudio Arroyo
(Orientador)

Dedico essa dissertação ao meu esposo
Christien

Aos meus pais
Maria do Carmo e Nominato

AGRADECIMENTOS

Primeiro agradeço a Deus pela força nos momentos difíceis, quando as coisas não davam certo, uma nova chance parecia possível.

Agradeço ao Christien pela ajuda em todos os momentos, pela paciência, pelo incentivo, pelo carinho e por acreditar em mim, em momentos que nem eu mesma acreditava.

Agradeço ao Professor José Elias Cláudio Arroyo, pelas inúmeras aulas, paciência em explicar tudo o que deveria aprender e por me encaminhar e mostrar a direção que deveria seguir.

Aos professores do Departamento de Informática, da Universidade Federal de Viçosa, pela formação e conhecimentos recebidos.

Aos colegas de mestrado que me incentivaram e contribuíram de forma direta ou indiretamente para realização desse trabalho. De forma especial aos amigos Alexandre Fraga e ao João Paulo Nogueira.

BIOGRAFIA

ANA AMÉLIA DE SOUZA PEREIRA, filha de Nominato Pereira e Maria do de Souza Moreira, brasileira nascida em 19 de julho de 1977, na cidade de Ubá, no estado de Minas Gerais.

No segundo semestre 2008, ingressou no programa de pós graduação em Ciência da Computação do Departamento de Informática – DPI, onde em agosto deste mesmo ano, iniciou o mestrado na Universidade Federal de Viçosa – UFV, defendendo sua dissertação em 02 de agosto de 2011.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVEATURAS E SIGLAS

AG – Algoritmo Genético

AG-B – Algoritmo genético básico

AG-BL – Algoritmo genético com busca local

CPLEX – Software matemático de otimização

EDD (*Earliest due date first*) – Menor data de entrega

ERD (*Earliest release date first*) – Menor data de liberação

GRASP – *Greedy randomized adaptive search procedure*

ILS – *Iterated local search*

JIT – *Just-in-time*

LPT (*Longest processing time*) – Maior tempo de processamento

MDD - *Modified due date*

MST (*Minimum slack time*) – Menor tempo de folga

NEH – Algoritmo de Nawaz, Enscore Jr. e Ham

NP – Classe de problemas de tempo polinomial não determinístico

PLIM – Programação linear inteira mista

PLI– Programação linear inteira

PRTT – *Priority rule for total tardiness*

RPD (*Relative percentual deviation*) – Desvio relativo percentual

SBOX – *Similar block order crossover*

SJOX – *Similar job order crossover*

SPT (*Shortest processing time*) – Menor tempo de processamento

WLPT (*Weighted longest processing time*) - Menor tempo de processamento ponderado

WSPT (*Weighted shortest processing time*) - Maior tempo de processamento ponderado

RESUMO

PEREIRA, Ana Amélia de Souza, M.Sc., Universidade Federal de Viçosa, Agosto de 2011. **Metaheurísticas para o problema de *flowshop* flexível com penalidades de adiantamento e atraso.** Orientador: José Elias Claudio Arroyo.

Este trabalho aborda o problema da programação de tarefas num sistema *flowshop* flexível, com o objetivo de minimizar penalidades por adiantamentos e atrasos em relação às datas de entrega das tarefas. Considera-se que todas as tarefas estão disponíveis para processamento em diferentes instantes, conforme suas datas de liberação (*release time*), além disso, existem tempos de preparação (*setup times*) dependentes da sequência e dos estágios. O problema abordado pertence ao ambiente de programação da produção JIT (*just-in-time*). Neste ambiente, o processamento de uma tarefa deve ser finalizado o mais próximo possível da sua data de entrega. Quando ocorre o adiantamento no processamento da tarefa, a produção é finalizada antes da data de entrega, devendo então ser armazenada, o que gera custos de estocagem. Já com o atraso na cadeia produtiva, ocorre o custo de atraso da tarefa e consequente atraso das demais produções, ocasionando um efeito em cadeia, além da perda de confiança por parte do consumidor, fator este de custo inestimável, além do pagamento de multas contratuais, o que prejudica a imagem de uma empresa. Neste trabalho foi desenvolvido um modelo de programação inteira mista (PLIM) para o problema. Devido à complexidade computacional do problema, são utilizadas heurísticas e metaheurísticas para a obtenção de soluções aproximadas de boa qualidade. As metaheurísticas aplicadas são Iterated Local Search (ILS) e Algoritmo Genético (AG): Algoritmo Genético básico (AG-B) e Algoritmo Genético com busca local (AG-BL). O AG-BL é uma adaptação do algoritmo genético básico com busca local, aplicada para melhorar soluções determinadas pelos operadores genéticos. O modelo de PLIM, para problemas de pequeno porte, é resolvido utilizando o software de otimização CPLEX. Os resultados das metaheurísticas propostas ILS, AG-B e AG-BL são comparados e analisados entre si e também com as soluções determinadas por duas heurísticas construtivas, MST e NEH-MST. Após os ajustes dos parâmetros necessários e diversos

testes, observou-se que a metaheurística ILS foi à técnica mais eficiente para obter soluções de boa qualidade para o problema estudado.

ABSTRACT

PEREIRA, Ana Amélia de Souza, M.Sc., Universidade Federal de Viçosa, August, 2011. **Metaheuristics for the flexible *flowshop* problem with earliness and tardiness penalties.** Advisor: José Elias Claudio Arroyo.

This work deals with the problem of job scheduling in flexible flowshop system with the objective of minimizing the penalties for tardiness and earliness in terms of the job due date. It is considered that all jobs are available for processing in different moments, according to them release times, besides that, there are setup times depending on the sequence and stages. The problem addressed belongs to the production programming environment JIT (just-in-time). In this environment, each job has a processing time and due date within which it should preferably be completed, therefore the processing a job should end as closer as possible to its due date. When a job finishes processing earlyth, production is finalized before due date, hence it must be stored, which generates storage costs. When the job is tardy is the job and consequent delay on further productions, causing a chain effect, causing loss of trust from consumer, which has unestimated cost, besides contract fines, which harms the image of an enterprise. In this work it was developed an PLIM for the problem. Due to the problem computational complexity, heuristics and metaheuristic are used for obtaining good quality approximate solutions. Metaheuristics applied are Iterated Local Search (ILS) and Genetic Algorithm (GA): basic genetic algorithm (GA-B) and Genetic Algorithm with local search (GA-LS). The GA-LS is an adaptation of the GA-B with local search, applied for improving a solutions determined by the genetic operators. PLIM model, for of problems smaller size, is solved by using the optimization software CPLEX. Results obtained from the proposed heuristics ILS, AG-B and AG-BL are compared and analyzed among themselves and also with solutions determined by constructive heuristics, MST and NEH-MST, computational tests have shown that metaheuristic ILS obtains solutions of good quality. After adjusting the required parameters and several tests showed that the ILS was the most effective technique for obtaining good quality solutions to the problem studied.

CAPÍTULO 1

INTRODUÇÃO

Com o advento da revolução industrial, na segunda metade do século XVIII, o mundo presenciou também o crescimento e complexidade das organizações, que perdura até os dias atuais. Como parte dessa mudança, destacou-se a divisão do trabalho e a segmentação da gerência, juntamente com a criação de novos problemas como: a organização do processo produtivo e a busca por um processo eficiente com mais lucro, que fazem parte da vida destas organizações na sociedade hodierna.

No contexto da nova problemática, o que é melhor para uma indústria, pode ser inadequada para a outra. Desta forma, a complexidade e especialidades de problemas também cresceram e se desenvolveram. Com tais mudanças, tornou-se mais difícil alocar as tarefas disponíveis para as diversas atividades, de forma que pudessem ser mais eficazes para a organização como um todo.

Tais problemas e a necessidade de encontrar uma melhor maneira de resolvê-los, favoreceu um ambiente para um novo campo: a Pesquisa Operacional (HILLIER & LIEBERMAN, 2004).

A técnica de Pesquisa Operacional teve seu início de desenvolvimento na Inglaterra, durante a Segunda Guerra Mundial (1939–1945). Foi utilizada para maximizar recursos escassos em operações militares, sendo que após a guerra, suas aplicações como: logística, tática, atualização tecnológica e padronização e otimização do processo produtivo, foram inseridas em empresas da Inglaterra e dos Estados Unidos (PEINADO & GRAEML, 2007).

A Pesquisa Operacional é utilizada, portanto, para analisar sistemas complexos do mundo real. Dentre os problemas que podem ser resolvidos utilizando esta análise, destacam-se os problemas de otimização combinatória.

A otimização combinatória é a disciplina do processo de decisão, que faz interface entre a matemática, a ciência da computação e a pesquisa operacional.

Os problemas de otimização combinatória surgem de situações nas quais algumas escolhas devem ser realizadas, resolvê-las equivale a encontrar uma solução ótima entre um número finito ou infinito de alternativas contábeis (AARTS & LENSTRA, 2003). Tais problemas são classificados como NP-difícil, ou ainda não são conhecidos algoritmos que os resolvam em tempo polinomial.

Problemas complexos envolvem uma seleção de valores para um número de variáveis interrelacionadas, com foco em um único objetivo: quantificar e medir a qualidade da decisão. Este objetivo é maximizado (ou minimizado) através de uma função, cujas variáveis devem obedecer a certas restrições. No mundo real existe uma variedade de problemas que podem ser resolvidos utilizando métodos de otimização combinatória (LUCENA & PONTES, 2007). Conforme Arenales *et al*. (2007), alguns problemas utilizados na prática:

- problemas de planejamento de programação (*scheduling*) da produção;
- problemas de corte e empacotamento;
- problemas de transporte, transbordo e designação;
- problemas de programação de projetos;
- problema de gestão financeira;
- problemas de meio ambiente;
- dentre outros.

A otimização combinatória é aplicada com sucesso para resolução de problemas de programação de produção, através da utilização de regras ótimas para ambientes que envolvem múltiplas tarefas, máquinas com diferentes medidas de desempenho e estações de trabalho.

Para estes problemas que pertencem a classe NP-difícil, um grande número de soluções como heurísticas e metaheurísticas são criadas no intuito de solucioná-los (KOGAN & KHMELNITSKY, 2000).

1.1 Motivação

De acordo com Peinado & Graeml, (2007) existe uma preocupação constante das empresas em manterem a qualidade aliada à produtividade, de forma a atender uma

clientela exigente em qualidade de produtos e serviços. As empresas estão sempre à procura de novas técnicas, tecnologias e métodos, para alcançar estes objetivos e ao mesmo tempo eliminar atividades que não agregam valor ao processo produtivo.

Outro fator importante que preocupa os administradores é a competitividade de mercado, diversas empresas fabricam produtos similares e para ganhar a concorrência precisam obter maior eficiência no processo produtivo, com menos produtos estocados, menor custo de produção, menor preço de venda, e entregas dentro do prazo. Desta forma, garantir a satisfação do cliente, a fim de ganhar a concorrência.

O gerenciamento eficiente das etapas da cadeia produtiva poderá proporcionar: estocagem reduzida, produtos com preços compatíveis com o mercado e satisfação do cliente em relação a preço e prazo de entrega. De forma a tornar a gestão da produção eficaz para alcançar os objetivos estratégicos da empresa.

A técnica *just-in-time* (JIT) é considerada uma ferramenta importante nas mãos dos gestores para alcançar os objetivos mencionados no parágrafo anterior.

Com a utilização da técnica JIT ocorre à otimização da utilização dos recursos materiais, de forma a alcançar o melhoramento da qualidade de produção e eficácia na entrega dos produtos (HUTCHINS, 1999). Através do *just-in-time*, desenvolve-se uma vantagem competitiva, para uma melhor administração de todo o sistema de manufatura. Portanto, para redução do desperdício, o gestor deve conhecer todo processo produtivo e eliminar paradas não programadas, diminuir tempos de preparação (*setup time*) e tempos de espera que não agregam valor ao produto.

Este trabalho aborda o problema da programação de tarefas num sistema *flowshop* flexível, com o objetivo de minimizar penalidades por adiantamentos e atrasos em relação às datas de entrega das tarefas. Este problema pertence ao ambiente de programação da produção JIT. Neste ambiente, o processamento de uma tarefa deve ser finalizado o mais próximo possível da sua data de entrega. Quando ocorre o adiantamento no processamento da tarefa, a produção é finalizada antes da data de entrega, devendo então ser armazenada, o que gera custos de estocagem. Já com o atraso na cadeia produtiva, ocorre o custo de atraso da tarefa e consequente atraso das demais produções, ocasionando um efeito em cadeia, além da perda de confiança por parte do consumidor, fator este de custo inestimável, além do pagamento de multas contratuais, o que prejudica a imagem de uma empresa.

No problema de programação *flowshop* flexível, estudado neste trabalho, existem n tarefas que devem ser processadas em k estágios em série. Cada estágio t possui m_t máquinas idênticas. Cada tarefa j possui uma data de entrega d_j e um tempo

de processamento P_{tj} no estágio t. Considera-se que todas as tarefas estão disponíveis para processamento em diferentes instantes conforme suas datas de liberação R_j (*release time*). Além disso, existem tempos de preparação (*setup times*) dependentes da sequencia e dos estágios.

1.2 Objetivos Gerais

Este trabalho possui como objetivos gerais testar o desempenho de algumas metaheurísticas, tais como ILS (*Iterated Local Search*) e AG (Algoritmo Genético), em busca de um melhor resultado para o problema de programação de tarefas no ambiente *flowshop* flexível com *setup time* dependente da sequência, minimizando respectivamente: adiantamento, atraso e tempo de finalização das tarefas.

1.3 Objetivos Específicos

Objetivos específicos desta dissertação são os seguintes:

a) Analisar um conjunto de casos do problema em busca de uma característica comum.
b) Implementar um modelo matemático do problema.
c) Implementar heurísticas construtivas, propostas na literatura, de modo a encontrar uma ordem de processamento das tarefas nas máquinas, com custo razoável.
d) Aplicar as metaheurísticas ILS e AG, comparar seus resultados obtidos e verificar se são satisfatórios.

1.4 Trabalhos Relacionados

Através de pesquisas realizadas nas bibliotecas digitais, foram encontrados vários artigos técnico-científicos que apresentam soluções voltadas para o problema de *flowshop* flexível. Alguns dos principais trabalhos sobre o assunto encontram-se organizados aqui.

Existem muitos trabalhos que tratam do problema de programação de tarefas no ambiente *flowshop* flexível com o objetivo de minimizar o *makespan* (tempo máximo de conclusão de todas as tarefas). Porém, poucos trabalhos abordam o problema com datas de entregas, *release time* das tarefas e *setup times* dependente da sequência.

Ruiz *et al*. (2006b) consideram a minimização do *makespan*, no problema de *flowshop* flexível, com as seguintes características: máquinas paralelas elegíveis em cada estágio (para cada tarefas são definidas as máquinas nas quais serão processadas), *release times* das máquinas (instante no qual a máquina estará disponível para ser usada), tempos de preparação dependentes da sequência e das máquinas. Eles propõem um modelo matemático de programação linear inteira (PLI) e para a resolução do problema foi utilizada uma adaptação da heurística NEH (NAWAZ *et al.*, 1983) e outras heurísticas construtivas baseadas em regras de prioridade.

Ruiz & Maroto (2006) propõem um AG com o objetivo de minimizar o *makespan*. Eles consideram máquinas paralelas independentes em cada estágio, máquinas elegíveis, tempos de preparação dependente de sequência para máquinas e estágios.

Kurz & Askin (2004) apresentam um AG e quatro heurísticas construtivas com objetivo de minimizar o *makespan* no ambiente *flowshop* flexível, com tempos de preparação dependente de sequência.

Mainieri & Ronconi (2008) abordam a minimização do atraso total no problema de *flowshop* flexível. Neste problema, o tempo de processamento de uma tarefa depende apenas da tarefa e do estágio, os tempos de preparação são independentes da sequência. Para resolver o problema, eles propõem uma adaptação da regra PRTT (*Priority Rule for Total Tardiness*) que consiste em minimizar o atraso total em um ambiente com uma única máquina e tarefas com diferentes tempos de liberação. Os resultados obtidos foram comparados com três heurísticas baseadas, respectivamente, nas regras SPT (*Shortest Processing Time*), EDD (*Earliest Due Date*) e MDD (*Modified Due Date*).

Jungwattanakit *et al.* (2008) consideram um ambiente de programação de tarefas com máquinas paralelas independentes nos estágios, datas de liberação e datas de entrega, tempos de preparação dependentes da máquina e da sequência. Nesta abordagem, o objetivo é minimizar o *makespan* e o atraso total das tarefas. Foram utilizadas heurísticas construtivas baseadas em regras de prioridade. Também é

proposto um AG no qual parte da população inicial é gerada utilizando as heurísticas construtivas.

Fakhrzad & Heydari (2008) abordam o problema de *flowshop* flexível com o objetivo de minimizar as penalidades totais por adiantamento e por atraso. Estes autores consideram máquinas paralelas idênticas, tempos de preparação independentes de sequência e desenvolvem um método de solução composto por três etapas. A primeira etapa aloca as tarefas às máquinas e transforma o problema de *flowshop* flexível em vários problemas simples de *flowshop* (problema com apenas uma máquina em cada estágio). Na segunda etapa, cada problema simples de *flowshop* é resolvido. Na terceira etapa, a partir das soluções obtidas na etapa anterior, constrói-se uma solução viável do problema. Esses autores também propõem um algoritmo híbrido utilizando a combinação das heurísticas *simulated annealing* e *tabu search*.

Davoudpour & Ashrafi (2009) apresentam o problema de *flowshop* flexível com o objetivo de minimizar simultaneamente quatro critérios: penalidades por adiantamento, por atraso, por tempo de conclusão e datas de entrega. Nessa abordagem, os tempos de preparação são dependentes da sequência, cada estágio é formado por máquinas paralelas idênticas e as tarefas possuem diferentes datas de liberação. Para a resolução do problema foi utilizada a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP).

Neste trabalho, para resolver o problema abordado, desenvolveu-se metaheurísticas de busca, a primeira baseada na heurística *Iterated Local Search* (ILS), uma em AG básico e outra em AG híbrido. O AG básico neste trabalho é chamado de AG-B e o AG híbrido chamado de AG-BL. Segundo Lourenço *et al.,* (2003), o ILS é um método heurístico de busca local muito utilizado na resolução de problemas de otimização combinatória. Abordagens do problema de *flowshop* flexível utilizando a metaheurística ILS, não foram encontradas na literatura. Já AG é um método evolutivo e também bastante aplicado na literatura de problemas de escalonamento de tarefas. Vale ressaltar que, o AG-BL utiliza uma heurística de busca local para melhorar soluções determinadas pelos operadores de *crossover* e mutação.

Nas próximas seções apresentam-se a representação de uma solução, como são geradas as soluções iniciais e detalhes da implementação dos algoritmos ILS, AG-B e do AG-BL.

1.5 Estrutura da dissertação

Este trabalho foi embasado em informações publicadas na literatura, através de pesquisa bibliográfica em trabalhos científicos relacionados ao contexto do problema abordado e da metodologia proposta.

O Capítulo 2 é apresentado referencial teórico, através da definição e descrição do problema.

A descrição do problema *flowshop* flexível está contida no Capítulo 3, onde é apresentado um modelo matemático de programação linear inteira mista (PLIM) do problema, composto pela função objetivo e suas restrições.

No Capítulo 4 estão expostos detalhes dos métodos de solução utilizados neste trabalho. É ilustrado um exemplo de sequenciamento de tarefas para o problema *flowshop* flexível, são descritos os métodos construtivos e as metaheurísticas propostas: MST, NEH-MST, ILS, AG-B e AG-BL. Todos estes métodos são exibidos, juntamente com o seu respectivo pseudocódigo.

Os resultados da aplicação das metaheurísticas propostas são apresentados no Capítulo 5. São detalhados, também, os parâmetros usados pelos métodos.

Finalmente no Capítulo 6, as conclusões obtidas, as principais contribuições e sugestões para trabalhos futuros são disponibilizadas.

CAPÍTULO 2

REFERENCIAL TEÓRICO

Neste capítulo é realizada uma revisão de literatura, abordando maiores detalhes sobre o caso de estudo neste projeto. Primeiramente será tratado o problema de programação de tarefas, logo em seguida, uma descrição sobre o ambiente *flowshop* flexível, juntamente com um exemplo de sua aplicação. Serão abordados também, conceitos de tempo de preparação, problema de adiantamento e atraso e finalmente, otimização combinatória com foco em heurísticas construtivas e metaheurísticas.

2.1 Problema de programação de tarefas

A programação da produção é utilizada por diferentes tipos de empresas, e tem como objetivo transformar as entradas em saídas desejadas. Estas entradas podem ser: produtos finalizados em outro sistema e ainda não acabados, matéria prima, entre outros (CHASE *et al*., 2006).

Neste trabalho, aborda-se o problema de programação de tarefas em máquinas. Segundo Pinedo (2008) a programação (sequenciamento) de tarefas é um processo decisório utilizado no setor de fabricação das indústrias. Programação de tarefas envolve alocação de recursos com tempos determinados e tem como função otimizar um ou mais objetivos. Os recursos podem ser máquinas em um estágio, ou operações de tarefas em um processo produtivo, sendo que cada uma pode ter um determinado grau de prioridade visando sua data de entrega e os objetivos podem assumir diferentes formas: minimizar tempos de conclusão ou minimizar atrasos com relação a datas de entrega.

Nos problemas de programação de tarefas, especificam-se as tarefas e os objetivos de programação, classificados em função do fluxo das tarefas nas máquinas e nos estágios, conforme segue (MACCARTHY & LIU, 1993):

- *Job Shop* – cada tarefa tem sua própria ordem de processamento;
- *Flowshop* – para todas as tarefas existe a mesma sequencia de processamento;
- *Open Shop* – não há sequência específica para as tarefas que serão processadas;
- Máquina Única – existe somente uma máquina para as tarefas serem processadas;
- Máquinas Paralelas – mais de uma máquina, geralmente idênticas, são disponibilizadas para as mesmas operações;
- *Job Shop* com Múltiplas Máquinas – *Job shop* com um conjunto de máquinas paralelas em cada estágio de produção;
- *Flowshop* com Múltiplas Máquinas – é um tipo de *flowshop* onde existe um conjunto de máquinas paralelas em cada estágio de produção;
- *Flowshop* Flexível – uma extensão do *flowshop* e de máquinas paralelas, caracterizado por estágios, cada estágio é composto de um conjunto de máquinas paralelas idênticas. Em um estágio, uma tarefa pode ser processada em qualquer máquina.

Nestes problemas também podem ser abordados diferentes medidas de desempenho, sendo que algumas das mais utilizadas são (SOUZA, 2009):

- *Makespan* – tempo total da programação de tarefas;
- *Mean flow-time* – tempo médio de duração do fluxo;
- *Total flow*-time – tempo total de duração do fluxo;
- *Mean tardiness* – atraso médio do fluxo de tarefas;
- *Maximum tardiness* – máximo do atraso na conclusão das tarefas;
- Soma das penalidades de atraso (*tardiness*) e adiantamento (*earliness*).

2.1.1 Programação de tarefas em um ambiente *flowshop* flexível

A programação da produção é uma decisão importante a ser tomada no processo produtivo, nos ambientes industriais modernos, onde problemas de escalonamento ocorrem com frequencia.

No problema de programação *flowshop* todas as tarefas possuem o mesmo roteiro, ou seja, devem ser processadas na mesma ordem do primeiro ao último estágio (MAINIERI, 2009).

Neste trabalho, será estudado o ambiente *flowshop* com múltiplos processadores, o qual é uma generalização do *flowshop*. Este ambiente também é chamado de *flowshop* flexível ou de *flowshop* híbrido.

No ambiente *fowshop* flexível existem máquinas em paralelo distribuídas em estágios, nos quais todas as tarefas devem passar em todos os estágios, apenas uma vez, em uma mesma ordem do primeiro ao último estágio.

A maioria dos problemas de programação são muito significativos e de difícil solução, devido à sua natureza complexa. Um exemplo deste tipo de ambiente pode ser visto na indústria têxtil, produtora de roupas, cintos, e na indústria de pneus, que possuem processos produtivos diversificados dependendo do tipo de produto a ser fabricado (JUNGWATTANAKIT et al., 2008).

O ambiente *flowshop* flexível é caracterizado pela existência de n tarefas e k estágios em série e alguns estágios podem ter apenas uma máquina, mas pelo menos um estágio deve ter várias máquinas e todas as tarefas deverão passar por todos os estágios na mesma ordem (MAINIERI; RONCONI, 2008). Uma máquina pode processar no máximo uma tarefa de cada vez e uma tarefa pode ser processada por no máximo uma máquina de cada vez (JUNGWATTANAKIT *et al*., 2008).

Um exemplo de uma possível programação de cinco tarefas, no ambiente *flowshop* flexível, está ilustrado na Figura 1, onde as tarefas devem passar em cada estágio obedecendo à sequência de tarefas 1, 4, 5, 3, 2.

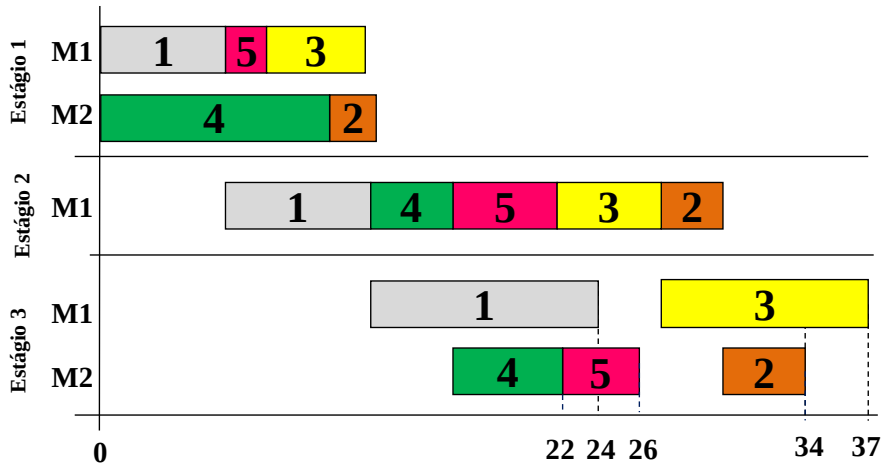

Figura 1- Exemplo de programação *flowshop* flexível

O instante de conclusão das tarefas 1, 2, 3, 4, 5 são respectivamente 24, 34, 37, 22 e 26.

2.1.2 Tempos de preparação (*Setup time*)

O tempo de preparação é o tempo gasto para organizar: a máquina, o processo, a matéria prima a ser transformada e o ciclo produtivo. Isto inclui obtenção de ferramentas, limpeza, trabalho de posicionamento no processo de material, realocação de ferramentas, ajuste de ferramentas, inspeção de materiais, escolha dos gabaritos necessários e utensílios (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999).

Diversos trabalhos que abordam problemas de programação de tarefas consideram o uso de tempos de preparação não significativos, ou então, incluem este tempo de preparação nos tempos de processamento das tarefas. A construção desta abordagem na prática é mais simplificada, mas pode afetar negativamente a qualidade das soluções geradas pelos aplicativos de programação. Pois estes tempos possuem uma variabilidade relevante em relação à ordenação das tarefas nas máquinas ou em relação a casos de antecipação de tarefas em máquinas, quando não há necessidade de aguardar o término da tarefa anterior (ALLAHVERDI *et al.*, 2008).

Existem basicamente dois tipos de problemas para tempos de preparação, separados dos tempos de processamento das tarefas. No primeiro, o tempo de preparação depende somente da tarefa que será executada, neste caso é tratado como independente da sequência de execução da tarefa. No segundo caso, existe a dependência tanto da tarefa a ser executada, quanto da tarefa executada anteriormente

na mesma máquina, chamado de dependente de sequência (ALLAHVERDI & GUPTA; ALDOWAISAN, 1999).

Segundo Allahverdi et al.(2008), muitos estudos em relação à importância de tempos de preparação tem sido realizados. A motivação para tais estudos é a economia adquirida com a inclusão de tempos de preparação, nas decisões de programação de produção (NADERI; ZANDIEH & FATEMI GHOMI, 2009).

2.1.3 Adiantamento (*Earliness*) e atraso (*Tardiness*)

O foco deste estudo deste trabalho é a minimização dos atrasos (*tardiness*) e adiantamento (*earliness*) das tarefas, com a inclusão de penalidades para as tarefas que não finalizarem em suas datas de entrega combinadas com o cliente.

O produto finalizado em horário diferente da data de entrega programada, faz com que seu custo aumente. No caso da finalização do produto ocorrer antes da data de entrega, poderá ocorrer custos em relação a espaço para estocagem do produto acabado, movimentação deste produto em estoque, produtos perecíveis poderão estragar e também outros custos secundários. Estes custos vindos do adiantamento do processamento de um conjunto de tarefas serão agrupados como um único coeficiente chamado de penalidade de adiantamento (COLIN, 1997).

Em contrapartida, os produtos podem também ser finalizados após a data de entrega programada e gerar problemas junto ao cliente, o qual poderá pedir descontos para ficar com o produto, cobrar uma multa pelo atraso na entrega do produto, ou até mesmo deixar de comprar da empresa por não cumprir com o combinado pré estabelecido. Neste caso, os custos envolvidos serão agrupados em um único coeficiente chamado de penalidade de atraso (COLIN, 1997).

A melhor sequência para a programação de tarefas é aquela onde as tarefas finalizam exatamente em suas datas de entrega, o que não ocorre sempre. Este tipo de problema gera um grande número de possibilidades de soluções, que aumenta rapidamente com o acréscimo de quantidades de tarefas, inviabilizando assim a escolha de uma melhor alternativa em termos de tempos computacionais (SOUZA, 2009).

Uma solução para este tipo de problema é a utilização de algoritmos heurísticos, os quais podem não atingir uma sequência ótima, mas encontram boas soluções em termos de tempos computacionais aceitáveis.

2.2 Otimização Combinatória

A otimização combinatória envolve o processo de encontrar soluções para variáveis discretas, de forma que a solução ótima em relação a uma função de determinado objetivo é encontrada. Vários problemas de otimização de importância tanto prática como teórica são de natureza combinatória. Pode-se utilizar o exemplo do problema do caminho mais curto, como outros tantos encontrados no mundo real, como: encontrar um mínimo plano de custo para entrega de mercadorias aos clientes, melhor esquema de roteamento de pacotes de dados na internet, sequência ótima para tarefas que serão transformadas em uma linha de produção, entre outros (DORIGO; STUTZELE, 2004).

Em problemas combinatórios o espaço de soluções possíveis (candidatas, viáveis) é finito, apesar de ser extremamente grande e discreto (PAPADIMITRIOU; STEIGLITZ, 1998).

Conforme Dorigo & Stutzele (2004) os problemas de otimização combinatória, sejam de maximização ou minimização, estão associados a um conjunto de instâncias. O termo instância refere-se a problemas com valores específicos para todos os parâmetros.

Os problemas de otimização combinatória podem ser classificados, conforme segue (GAREY & JOHNSON, 1999):

- P (*Polinomial Time*) – são problemas que podem ser resolvidos por algoritmos polinomiais, estes problemas são consideráveis possíveis de solução de forma eficiente;
- NP (*NonDeterministic Polinomial Time*) – problemas que podem ser resolvidos por algoritmos não-determinísticos polinomiais. Nestes problemas a solução pode ser encontrada em tempo polinomial;
- NP-completo – é um subconjunto dos problemas de decisão em NP. São problemas que existe redução de tempo polinomial a partir de qualquer problema.
- NP-difícil – problemas que podem ser resolvidos por um número polinomial de soluções de um problema NP-completo.

O problema de programação *flowshop* flexível é um problema de natureza combinatória, que em termos de complexidade computacional é classificado como NP-

difícil. Desta forma, este tipo de problema só pode ser resolvido eficientemente, de maneira ótima, em casos de pequeno porte.

Para viabilizar a resolução destes problemas de otimização, como é o caso do problema de programação *flowshp* flexível, os métodos mais utilizados são os métodos heurísticos.

No caso dos métodos exatos, existe a procura por uma solução ótima para os problemas, onde são utilizados modelos matemáticos de otimização e algoritmos específicos. O que acontece em muitos casos, devido à complexidade combinatorial, é que há uma necessidade de alto tempo computacional, tornando inviável a busca por soluções ótimas.

Em contrapartida, os métodos heurísticos possibilitam encontrar soluções viáveis em tempos de execução polinomial, a diferença é que não garantem a obtenção da solução ótima do problema.

Para problemas de pequeno porte justifica-se a utilização dos métodos exatos. No caso de muitos problemas práticos, em geral maiores, deve-se levar em consideração o equilíbrio entre a qualidade da solução, a simplicidade, a implantação e a eficiência computacional destes problemas.

Nos problemas de programação de tarefas com datas de entrega, busca-se por tarefas que finalizem seu processamento o mais próximo possível de sua data de entrega pré estabelecida. A dificuldade existente para resolver estes problemas é o aumento rápido do número de soluções possíveis à medida que crescem os números de tarefas. Desta forma, justifica-se investir em métodos heurísticos, os quais tendem não irão garantir uma solução ótima, mas encontrarão soluções de boa qualidade, com tempos de processamentos aceitáveis (SOUZA, 2009).

As subseções 2.2.1 e 2.2.2 tratam das heurísticas construtivas e metaheurísticas, respectivamente.

2.2.1 Heurísticas Construtivas

De acordo com Gigante, (2010) os métodos heurísticos podem ser classificados em construtivos e melhorativos:

- Heurísticas Construtivas – Estes métodos constroem uma solução progressivamente. O algoritmo sempre começa com uma solução parcial

nula, e a cada passo, um elemento é adicionado à solução parcial, geralmente de forma gulosa. O algoritmo finaliza quando for construída uma solução completa, ou seja, uma solução que contém todos os elementos.

- Heurísticas Melhorativas – Estes métodos são procedimentos que tentam, iterativamente, melhorar uma solução por meio de perturbações, procura-se por uma solução melhor que a atual quanto à medida de desempenho adotada.

As soluções obtidas pelos métodos construtivos, geralmente, são utilizadas como soluções iniciais dos métodos de melhoria e/ou metaheurísticas (ARROYO, 2002).

2.2.2 Metaheurísticas

Conforme Osman & Kelly (1996), metaheurísticas são métodos aproximados, criados para resolver problemas de otimização combinatória, onde heurísticas clássicas não conseguem ser eficazes e eficientes. Metaheurísticas podem ser desenvolvidas através da combinação de diferentes conceitos derivados de heurísticas clássicas, inteligência artificial, redes neurais, entre outros.

Metaheurísticas são métodos genéricos que podem ser aplicados para resolver vários tipos de problemas obtendo bons resultados (COLLETTE & SIARRY, 2003). Quase todas as metaheurísticas propostas na literatura têm sido utilizadas para resolver problemas de *scheduling* (XHAFA & ABRAHAM, 2008).

Segundo Xhafa & Abraham (2008), o problema de *scheduling flowshop* é NP-difícil. O que justifica o uso de metaheurísticas para resolver esse problema.

Segundo Lucena & Pontes (2007), os métodos heurísticos são importantes na busca de soluções para problemas de natureza combinatória, permitindo a obtenção de resultados próximos do ótimo, ocorrendo algumas vezes destes serem ótimos.

Na busca por soluções heurísticas de boa qualidade, uma variedade de algoritmos de alto nível são desenvolvidos. Estes algoritmos podem ser específicos para resolver problemas de otimização combinatória e são conhecidos como metaheurísticas. As metaheurísticas são criadas a partir da observação de acontecimentos existentes no dia a dia, como o comportamento das formigas, pombos, atividades humanas, entre outras ações encontradas na natureza (LUCENA & PONTES, 2007).

Em relação à classificação das metaheurísticas, ainda não existe um consenso entre os autores, no que se refere a padrão de classificação. Algumas classificações importantes das metaheurísticas são:

- Metaheurística de métodos de relaxação – procedimentos para resolver problemas com modificações no modelo original, a solução gerada fornece a solução para o problema original. Ex.: relaxação lagrangeana.
- Metaheurística de busca em vizinhança – procedimentos que percorrem espaços de busca, onde se deve considerar em cada passo, a vizinhança da solução obtida na iteração anterior. Ex.: GLS (*Guided Local Search*), *Simulated Annealing, Tabu Search*, GRASP (*Greedy Randomized Adaptive Search Procedure*), *Iterated local search*, VNS (*Variable Neighborhood Search*).
- Metaheurística baseadas em métodos evolutivos – procedimentos focados em conjuntos de soluções que evoluem neste espaço. Ex.: Algoritmos genéticos, Estimação de distribuição, *Scatter Search*, *Ant colony optimization*, *Particle swarm optimization*.
- Metaheurística híbrida – procedimentos que combinam duas ou mais metaheurísticas e usam estratégias de busca tais como *Path Relinking.*

São procedimentos classificados como metaheurísticas, por proporcionarem uma melhora na heurística e fugirem dos ótimos locais.

CAPÍTULO 3

DESCRIÇÃO DO PROBLEMA *FLOWSHOP* FLEXÍVEL

Neste capítulo é apresentado um modelo matemático de programação linear inteira mista, para minimização da penalidade por adiantamento e por atraso, em um problema de programação *flowshop* flexível, com tempos de preparação (*setup times*) dependentes da sequência e dos estágios.

3.1 Modelagem matemática do problema

O problema de *flowshop* flexível é cada vez mais popular nas indústrias, principalmente em casos onde existe o aumento da carga de trabalho por estágios. Nestes casos é necessário mais máquinas em cada estágio, caracterizando máquinas do *flowshop* flexível (LOGENGRAN *et al.*, 2006).

Este ambiente é aqui abordado, tendo como característica o processamento de tarefas em linha de produção, onde n tarefas são processadas em k estágios na mesma ordem. Cada estágio t conta com m_t máquinas idênticas e uma tarefa pode ser processada por qualquer máquina do estágio corrente. São considerados tempos de preparação (*setup times*) das máquinas que dependem da sequência de tarefas, ou seja, quando a máquina finaliza o processamento de uma tarefa e antes de iniciar o processamento de outra tarefa, se gasta um tempo para preparar/ajustar a máquina. Cada tarefa possui uma data de entrega e deve-se levar em consideração que as tarefas estão disponíveis/liberação para sua execução em diferentes tempos (*release time*).

Neste trabalho foi desenvolvido um modelo matemático de programação linear inteira mista (PLIM). Este modelo é uma adaptação do modelo proposto por Fakhrzad & Heydari (2008). A seguir apresenta-se a descrição do modelo de PLIM.

Índices:

j, l: índices para tarefas, $j, l = 0,1..., n$;

t: índice para estágios, $t = 1,..., k$;

i: índice para máquinas, $i = 1,..., m_t$.

Parâmetros:

n: número de tarefas;

k: número de estágios;

m_t: número de máquinas em cada estágio t;

P_{tj}: tempo de processamento no estágio t da tarefa j;

d_j: data de entrega da tarefa j;

R_j: instante de liberação (*release time*) da tarefa j;

S_{tjl}: tempo de preparação no estágio t da tarefa j para tarefa l;

h_j: penalidade de adiantamento da tarefa j;

w_j: penalidade de atraso da tarefa j.

Variáveis:

C_{tj}: instante de conclusão no estágio t da tarefa j;

T_j: atraso da tarefa j;

E_j: adiantamento da tarefa j;

X_{jlti} { 1, se a tarefa l designada após a tarefa j no estágio t na máquina i;
0, caso contrário

O critério de desempenho utilizado é a minimização da penalidade total por atraso e por adiantamento, com relação às datas de entrega das tarefas. A seguir é apresentada a formulação matemática do problema.

Formulação:

$$\textit{Minimizar} \quad f = \sum_{j=1}^{n} h_j E_j + \sum_{j=1}^{n} w_j T_j, \tag{1}$$

Sujeito a:

$$\sum_{i=1}^{m_t} \sum_{j=0}^{n} X_{jlti} = 1; \quad l = 1,\ldots,n;\ t = 1,\ldots,k \tag{2}$$

$$\sum_{l=1}^{n+1} X_{jlti} \leq 1; \quad j = 0,\ldots,n;\ t = 1,\ldots,k;\ i = 1,\ldots,m_t \tag{3}$$

$$\sum_{l=1}^{n+1} X_{jlti} - \sum_{l=0}^{n} X_{ljti} = 0; \quad j = 1,\ldots,n;\ t = 1,\ldots,k;\ i = 1,\ldots,m_t \tag{4}$$

$$X_{jjti} = 0;\ j = 1,\ldots,n; \quad t = 1,\ldots,k;\ i = 1,\ldots,m_t \tag{5}$$

$$C_{tl} - C_{tj} \geq S_{tjl} + P_{tl} \times \sum_{i=1}^{m_t} X_{jlti} + M \times \sum_{i=1}^{m_t} X_{jlti} - 1; \quad j = 1,\ldots,n; \tag{6}$$

$$l = 1,\ldots,n;\ j \neq 1;\ t = 1,\ldots,k$$

$$C_{t+1\,j} - C_{tj} \geq P_{(t+1)\,j}; \quad j = 1,\ldots,n;\ t = 1,\ldots,k\text{-}1 \tag{7}$$

$$C_{1j} \geq R_j + P_{1j}; \quad j = 1,\ldots,n \tag{8}$$

$$C_{0\,j} \geq R_j; \quad j = 1,\ldots,n \tag{9}$$

$$C_{tj} \geq 0; \quad j = 0,\ldots,n;\ t = 0,\ldots,k \tag{10}$$

$$E_j \geq d_j - C_{tj}; \quad j = 1,\ldots,n \tag{11}$$

$$E_j \geq 0; \quad j = 1,\ldots,n \tag{12}$$

$$T_j \geq C_{tj} - d_j; \quad j = 1,\ldots,n \tag{13}$$

$$T_j \geq 0; \quad j = 1,\ldots,n \tag{14}$$

A equação (1) representa a minimização da soma das penalidades de atraso e adiantamento das tarefas. A restrição (2) garante que apenas uma tarefa *j* preceda uma tarefa *l* no estágio *t* numa máquina *i*. Já a restrição (3) garante que no máximo uma tarefa *l* suceda uma tarefa *j* em um estágio *t* na máquina *i*. A restrição (4) indica que cada tarefa tenha uma antecessora e uma sucessora no estágio *t* na máquina *i* na qual ela seja programada. A restrição (5) garante que uma tarefa não seja precedida por ela mesma. A restrição (6) indica que o tempo de conclusão da tarefa *j* que precede imediatamente a tarefa *l* seja maior ou igual à soma de seus tempos de processamento mais o tempo de preparação (*setup time*) da tarefa em todas as máquinas *i* do estágio *t*. A constante *M* representa um número suficientemente grande, que deve ser definido com base nos tempos de conclusão das tarefas. Já a restrição (7) garante a precedência do processamento das tarefas entre os estágios, ou seja, uma tarefa não pode ser processada no estágio $t+1$ antes de ter sido processada no estágio *t*. A restrição (8) garante que o instante de conclusão de uma tarefa no primeiro estágio, seja maior ou igual à soma de sua data de liberação mais seu tempo de processamento neste estágio. A restrição (9) aplica-se somente no primeiro estágio, onde uma tarefa não pode iniciar o seu processamento antes de sua data de liberação. Nas restrições (11) e (12) são calculados os adiantamentos das tarefas e nas restrições (13) e (14) são determinados os atrasos.

CAPÍTULO 4

MÉTODOS DE SOLUÇÃO

Conforme Jungwattanakit *et al*. (2008), o problema de programação *flowshop* flexível pertence à classe NP-difícil. Nos problemas de programação da produção, o fator tempo exerce uma importante influência no processamento das tarefas.

O método a ser utilizado neste tipo de problema precisa ser eficiente para determinar a melhor sequência de processamento, pois um método lento pode causar prejuízos diretos e indiretos a uma indústria. Para que uma tomada de decisão seja ágil, devem ser utilizados métodos baseados em heurísticas e metaheurísticas, ao invés de métodos de otimização exata. Heurísticas e metaheurísticas são métodos que determinam soluções aproximadas em tempo polinomial e são bastante utilizadas na resolução de problemas de otimização combinatória, NP-difíceis.

Com o objetivo de resolver o problema de programação de tarefas em um sistema *flowshop* flexível, neste trabalho é proposta uma adaptação da metaheurística ILS proposta por Dong *et al*. (2009), um AG básico e um AG híbrido que utiliza uma heurística de busca local para melhorar soluções determinadas pelos operadores genéticos. O AG básico é denominado como AG-B e o AG híbrido (com busca local) recebe a denominação de AG-BL.

Nas próximas seções apresentam-se, a representação de uma solução, como são geradas as soluções iniciais e detalhes da implementação dos algoritmos ILS, AG-B e do AG-BL.

4.1 Representação e avaliação de uma solução

Uma solução do problema é representada por uma permutação (sequência) π de n tarefas. A tarefa na posição j é denotada por $\pi(j)$, $j = 1,..., n$. Cada tarefa deve ser processada por apenas uma máquina i em cada estágio t, onde $t \in k$ e $i \in m_t$. Uma solução é avaliada pelo valor da função objetivo representada na equação (1) do modelo matemático; para tanto, é necessário calcular o tempo de finalização das tarefas no último estágio.

Para cada tarefa $\pi(j)$, calcula-se seu tempo de conclusão no estágio t, usando a seguinte fórmula recursiva (RUIZ & MAROTO, 2006):

$$C_{t,\pi(j)} = \min_{i=1}^{m_t} \{\max\{C_{t,L_{t_i}} + S_{t_i,L_{t_i},\pi(j)}; \max\{C_{t-1,\pi(j)}, R_{\pi(j)}\}\} + P_{t_i,\pi(j)}\} \quad (15)$$

Onde:

L_{t_i} : é a última tarefa processada na máquina i do estágio t;

$C_{t,L_{t_i}}$: tempo de conclusão no estágio t da tarefa L_{t_i} ;

$S_{t_i,L_{t_i},\pi(j)}$: *setup time* da máquina i do estágio t depois de processar a tarefa L_{t_i} e antes de processar a tarefa $\pi(j)$;

$C_{t,L_{t_i}} + S_{t_i,L_{t_i},\pi(j)}$: tempo de início do processamento da tarefa $\pi(j)$ se ela for inserida na máquina i do estágio t;

$C_{t-1,\pi(j)}$: tempo de conclusão no estágio anterior (t-1) da tarefa $\pi(j)$;

$R_{\pi(j)}$: instante de liberação da tarefa $\pi(j)$;

$P_{t_i,\pi(j)}$: tempo de processamento no estágio t na máquina i da tarefa $\pi(j)$.

Na Tabela 1, apresentam-se os dados de entrada para uma instância com 5 tarefas e 3 estágios, sendo o primeiro e o terceiro estágio, compostos por 2 máquinas e o segundo por uma máquina. Também são apresentados os tempos de processamento de cada tarefa, suas datas de entrega, seu instante de liberação, tempos de preparação e as penalidades por adiantamento e por atraso. O escalonamento das tarefas é ilustrado na

Figura 2. O exemplo descrito possui a sequência π = (1, 4, 5, 3, 2) com o valor da função objetivo de $f(\pi)$ = 617. As tarefas 1 e 4 são finalizadas no instante 34 e as tarefas 5, 3 e 2 nos instantes 46, 55 e 58 respectivamente.

Tabela 1 - Dados de entrada para uma instância do problema com 5 tarefas e 3 estágios

Parâmetros		**Nº Máquinas**	**Tarefa 1**	**Tarefa 2**	**Tarefa 3**	**Tarefa 4**	**Tarefa 5**
Tempo de processamento da tarefa j no estágio t (P_{tj})	Estágio 1	2	6	2	3	11	2
	Estágio 2	1	7	3	5	4	5
	Estágio 3	2	11	4	10	6	3
Data Entrega (d_j)			17	90	85	18	12
Instante de Liberação (R_j)			10	2	4	9	8
Penalidade de Adiantamento (h_j)			4	1	6	3	7
Penalidade de Atraso (w_j)			9	8	1	3	6
Setup Time (S_{tjl})	Estágio 1	Tarefa 0 (j)	0	0	0	0	0
		Tarefa 1 (j)	6	3	4	1	8
		Tarefa 2 (j)	3	5	2	4	3
		Tarefa 3 (j)	1	2	5	3	2
		Tarefa 4 (j)	7	1	8	5	3
		Tarefa 5 (j)	4	3	8	2	7
	Estágio 2	Tarefa 0 (j)	4	2	7	3	5
		Tarefa 1 (j)	2	3	4	1	8
		Tarefa 2 (j)	3	2	5	9	3
		Tarefa 3 (j)	6	4	2	5	4
		Tarefa 4 (j)	6	4	2	5	4
		Tarefa 5 (j)	1	9	3	2	7
	Estágio 3	Tarefa 0 (j)	5	8	6	1	3
		Tarefa 1 (j)	1	5	9	6	9
		Tarefa 2 (j)	3	4	1	7	8
		Tarefa 3 (j)	6	2	3	5	4
		Tarefa 4 (j)	1	9	3	2	7
		Tarefa 5 (j)	5	8	2	7	6

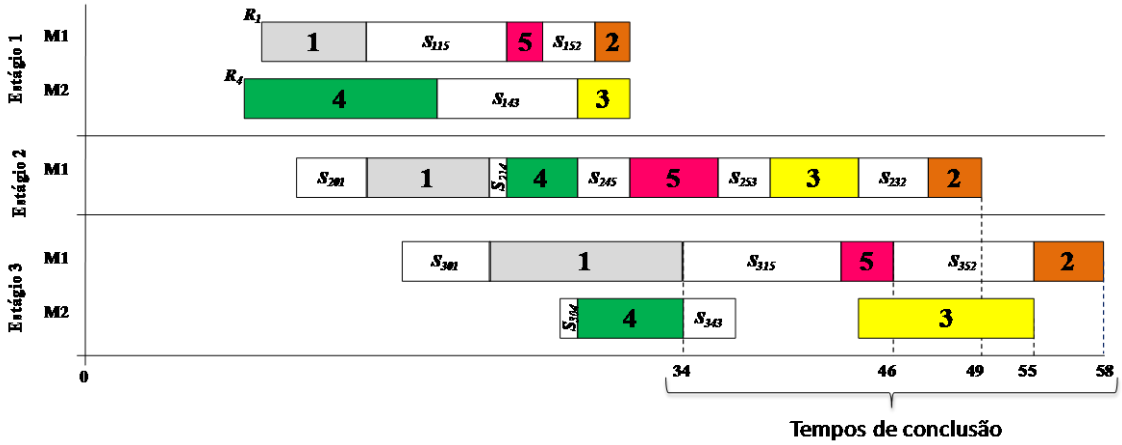

Figura 2 - Exemplo do escalonamento das tarefas com sequência π = (1,4,5,3,2)

Para demonstrar a utilização da fórmula recursiva, calcula-se passo a passo o tempo de conclusão da tarefa 2 no estágio 3 ($C_{3,2}$). Na sequência π = (1, 4, 5, 3, 2), supõe-se que, a tarefa 2 no estágio anterior 2 finalize no tempo 49, a tarefa 5 no estágio atual finalize no tempo 46 e a tarefa 3 finalize no tempo 55. Utilizando a Equação (15), $C_{3,2}$ é calculado da seguinte maneira:

ESTÁGIO 03
MÁQUINA 01

$$C_{3,2} = \min_{i=1}^{2_3}\{\max\{C_{3,5_{3_1}} + S_{3_1,5_{3_1},2}; \max\{C_{2,2}, R_2\}\} + P_{3_1,2}\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{\max\{46+8; \max\{52,2\}\} + 4\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{\max\{54;52\} + 4\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{54+4\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{58\}$$

ESTÁGIO 03
MÁQUINA 01

$$C_{3,2} = \min_{i=1}^{2_3}\{\max\{C_{3,3_{3_2}} + S_{3_2,3_{3_2},2}; \max\{C_{2,2}, R_2\}\} + P_{3_2,2}\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{\max\{55+2; \max\{52,2\}\} + 4\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{\max\{57;52\} + 4\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{57+4\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{61\}$$

$$C_{3,2} = \min_{i=1}^{2_3}\{58,61\} \longrightarrow C_{3,2} = 58$$

4.2 Gerações de soluções iniciais

Métodos de programação baseado em regras de prioridade são largamente utilizados em operações dentro dos ambientes de produção, em especial nos sistemas complexos como é o caso deste trabalho. A tarefa de maior prioridade é selecionada sempre que existe uma máquina disponível. Esta prioridade é definida por uma regra de despacho (MAINIERI, 2009).

Segundo Jungwattanakit *et al.* (2008), para determinar uma sequência de tarefas inicial, algumas regras de despacho tem sido amplamente utilizadas para problemas de *flowshop*, algumas delas encontram-se relacionadas abaixo: SPT, LPT, EDD, ERD e MST

A regra SPT (*Shortest Processing Time*) é uma regra de despacho simples, responsável por sequenciar as tarefas em ordem crescente dos seus tempos totais de processamentos:

$$\sum_{t=1}^{k} P_{t,\pi(j)} \tag{16}$$

Enquanto na regra LPT (*Longest Processing Time*) as tarefas são sequenciadas em ordem decrescente de seus tempos de processamento: $\sum_{t=1}^{k} P_{t,\pi(j)}$

Na regra EDD (*Earliest Due Date first*) as tarefas são sequenciadas em ordem crescente de suas datas de entrega, a qual tende a minimizar os atrasos.

Já na regra ERD (*Earliest Release Date first*) à ordenação é realizada pela ordem crescente da data de liberação, a qual tende a minimização da variação do tempo de espera no primeiro estágio.

A regra MST (*Minimum Slack Time*) é uma variação da regra EDD, definida pela sequência crescente da data de entrega de cada tarefa menos seus tempos de processamento (JUNGWATTANAKIT *et al.*, 2008):

$$d_j - \sum_{t=1}^{k} P_{t,\pi(j)} \tag{17}$$

Existem também outras regras que podem ser utilizadas para problemas de *flowshop* como: WSPT e WLPT.

Na regra WSPT (*Weighted Shortest Processing Time*) as tarefas são sequenciadas de acordo com a maior razão entre as penalidades de atraso e o tempo de processamento (COLIN, 1997):

$$\frac{w_{\pi(j)}}{\sum_{t=1}^{k} P_{t,\pi(j)}} \tag{18}$$

A ordenação na regra WLPT (*Weighted Longest Processing Time*), ocorre conforme a menor razão entre a penalidade de adiantamento e o tempo de processamento da tarefa (COLIN, 1997):

$$\frac{h_{\pi(j)}}{\sum_{t=1}^{k} P_{t,\pi(j)}} \tag{19}$$

Na Tabela 2 utiliza-se a mesma instância apresentada na Tabela 1, com o objetivo de demonstrar as sequências geradas pelas regras de despacho (prioridade): SPT, LPT, MST, EDD, ERD, WSPT e WLPT.

Tabela 2 - Exemplos das sequências geradas pelas regras de despacho

Tarefas			**T 1**	**T 2**	**T 3**	**T 4**	**T 5**
Tempo de processamento da tarefa *j* no estágio *t*	*Pt j*	Estágio 1	6	2	3	11	2
		Estágio 2	7	3	5	4	5
		Estágio 3	11	4	10	6	3
Data Entrega	*dj*		17	90	85	18	12
Instante de Liberação	*Rj*		10	2	4	9	8
Penalidade de Adiantamento	*hj*		4	1	6	3	7
Penalidade de Atraso	*wj*		9	8	1	3	6
Sequências geradas pelas Regras de Despacho	SPT		**T2**	**T5**	**T3**	**T4**	**T1**
	LPT		**T1**	**T4**	**T3**	**T5**	**T2**
	MST		**T1**	**T4**	**T5**	**T3**	**T2**
	EDD		**T5**	**T1**	**T4**	**T3**	**T2**
	ERD		**T2**	**T3**	**T5**	**T4**	**T1**
	WSPT		**T2**	**T5**	**T3**	**T1**	**T4**
	WLPT		**T2**	**T4**	**T1**	**T3**	**T5**

As gerações de sequências, através destas regras, requerem baixos tempos computacionais. As soluções obtidas por estas regras podem ser melhoradas por heurísticas de busca local e metaheuristicas.

Segundo Ruiz *et al*. (2006b), a heurística proposta por Nawaz *et al.* (1983) é um método eficiente para problemas de programação *flowshop* flexível, denotado por NEH. Este método é uma heurística gulosa de enumeração parcial, que constrói uma seqüência baseada na inserção de tarefas em ordem determinada.

Na Figura 3 é ilustrada a heurística NEH. Esta heurística inicia com uma solução parcial π formada pela primeira tarefa *j* da lista candidata *LC*. Uma solução completa é construída em *n*–1 iterações (i = 2,…,*n*). A cada iteração *i*, a próxima tarefa de *LC* é selecionada e inserida em todas as possíveis posições da atual sequência. A sequência gerada com o menor valor da função objetivo é escolhida.

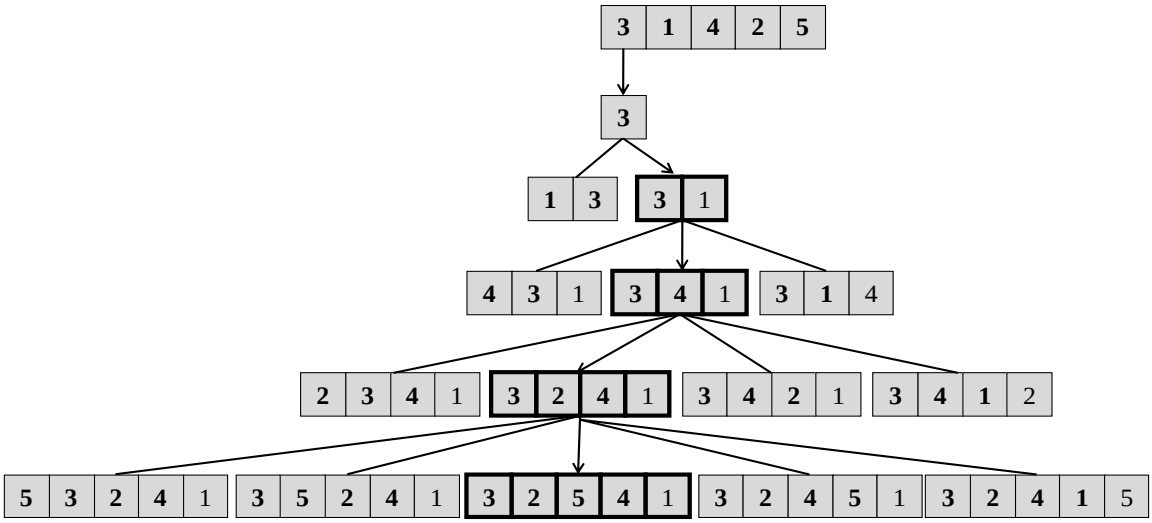

Figura 3 - Heurística construtiva NEH

Nessa heurística, inicialmente, as tarefas devem ser ordenadas de acordo com alguma regra de despacho. Neste trabalho, são utilizadas as seguintes regras para NEH:

SPT (*Shortest Processing Time*) – Menor tempo de processamento

LPT (*Longest Processing Time*) – Maior tempo de processamento

MST (*Minimum Slack Time*) – Menor tempo de folga

EDD (*Earliest Due Date first*) – Menor data de entrega

ERD (*Earliest Release Date first*) – Menor data de liberação

WSPT (*Weighted Shortest Processing Time*) - Maior tempo de processamento ponderado

WLPT (*Weighted Longest Processing Time*) - Menor tempo de processamento ponderado

Na heurística NEH implementada neste trabalho, ao invés de escolher a primeira tarefa da lista candidata *LC* para ser inserida em todas as posições da sequência parcial, escolhe-se aleatoriamente uma tarefa dentre as *h* primeiras tarefas da lista, $h = max(1, \alpha \times |LC|)$. Desta maneira, a heurística NEH, com uma regra de despacho, pode determinar soluções diferentes a cada execução. Note que, se $\alpha = 0$, sempre se escolhe a primeira tarefa da lista ordenada. Se $\alpha = 1$, a escolha da tarefa é aleatória dentre todas as tarefas. O pseudocódigo da heurística NEH é apresentado na Figura 4.

NEH
1. Ordenar as tarefas de acordo com uma regra de despacho obtendo a lista $LC := \{\pi(1),...,\pi(n)\}$;
2. Selecionar a tarefa $\pi(j)$ aleatoriamente de $LC = \{\pi(1)_{,...,}, \pi(h)\}$, onde $h := max(1, \alpha \times |LC|)$;
3. $\pi := \pi(j)$ uma sequência parcial formada pela tarefa selecionada;
4. Remover a tarefa $\pi(j)$ em *LC*;
5. **Para** *i* := 2 até *n* **faça**
6. Selecionar a tarefa $\pi(j)$ aleatoriamente de *LCR,* onde *LCR* é formada pelas primeiras $h := max(1, \alpha \times |LC|)$ tarefas de *LC*;
7. Insira a tarefa $\pi(j)$ em todas as posições possíveis de π, gerando *i* sequências parciais com *i* tarefas;
8. π' := Selecione a melhor sequência gerada;
9. Remover a tarefa $\pi(j)$ de *LC*;
10. $\pi := \pi'$;
11. **Fim – Para**
12. Retorne a sequência π com *n* tarefas.

Figura 4 - Pseudocódigo da heurística NEH

Neste pseudocódigo, a lista de candidatos *LC* é formada por um das regras propostas neste trabalho. A lista de candidatos restrita (*LCR*) é composta pelas primeiras $h = \alpha \times |LC|/100$ tarefas de *LC*. O algoritmo inicia com uma sequência parcial $\pi = \pi(j)$, formada por uma tarefa selecionada aleatoriamente em *LCR*. Uma solução completa é construída em *n*-1 iterações. A cada iteração *i* (*i*=2,...,*n*), uma tarefa $\pi(j)$ é selecionada aleatoriamente em *LCR* e é inserida em cada posição de π formando *i* sequências parciais com $\pi(j)$ tarefas.

4.3 Metaheurística ILS

O *Iterated Local Search* (ILS) é um algoritmo heurístico proposto por Lourenço *et al.* (2003). Esta heurística é baseada na idéia de que um procedimento de

busca local pode ser melhorado, gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações numa solução ótima local. A perturbação deve permitir que a busca local explore diferentes soluções e, além disso, deve evitar um reinício aleatório. O método ILS é, portanto, um método de busca local que procura focar a busca não no espaço completo de soluções, mas em um pequeno subespaço, definido por soluções que são ótimos locais.

O algoritmo ILS possui 4 etapas principais: obtenção de uma solução inicial π, perturbação da solução π obtendo uma solução π', melhoria da solução π' obtendo π'' (busca local) e um critério de aceitação da solução atual. Este critério pode aceitar soluções piores com o objetivo de sair de ótimos locais. As três últimas etapas do algoritmo são executadas iterativamente durante um número de vezes e algoritmo retorna a melhor solução obtida durante toda sua a execução.

Neste trabalho é proposta uma adaptação do algoritmo ILS de Dong *et al.* (2009) para resolver o problema *flowshop* flexível com penalidades por antecipação e atraso.

O pseudocódigo da metaheurística ILS é apresentado na Figura 5. No passo 2 é determinada uma solução inicial. Esta solução é obtida através de uma adaptação da heurística NEH. Usando duas regras de despacho: MST (*Minimum Slack Time*), e EDD (*Earliest Due Date first*).

A heurística NEH com a regra MST, neste trabalho, é chamada de NEH-MST e com a regra EDD é chamada de NEH-EDD.

A busca local utilizada no algoritmo ILS acontece nos passos 5, 6 e 7. Esta busca local consiste em inserir (ou realocar) uma tarefa $\pi(k)$ em cada uma das n-1 posições da sequência π atual, gerando n-1 sequências (vizinhas). A escolha da posição da tarefa $\pi(k)$ depende da ordenação da melhor sequência (solução) π*, encontrada até o momento (veja passo 5). Das sequências geradas escolhe-se a melhor para substituir a sequencia atual (passos 7 e 8). Se durante n iterações consecutivas da busca local não ocorrer melhora da sequencia atual, é ativado o procedimento de perturbação. A perturbação da solução π* é realizada no passo 12.

Os passos do procedimento da perturbação são descritos no pseudocódigo apresentado na Figura 6. Esta perturbação gera uma sequência π realizando um número *pert* de trocas aleatórias entre tarefas adjacentes da melhor sequencia π*. Neste trabalho foram testadas diferentes valores de perturbação, *pert* = 4, 6, 8, 10, 12 e 15.

```
ILS (pert, CondiçãoParada)
1.  iter := 1, cnt := 0;
2.  π := GerarUmaSoluçãoInicial( ); π* := π;
3.  Enquanto CondiçãoParada faça
4.      Para i := 1 até n faça
5.          Determine a posição k, tal que π(k) := π*(i);
6.          Insira a tarefa π(k) em cada uma das n-1 posições de π;
7.          Seja π' a melhor sequência, dentre as n-1 sequências geradas;
8.          Se f(π') < f(π) então π := π', cnt := 0;
9.          Senão cnt := cnt + 1; Fim – Se
10.         Se f(π) < f(π*) então π* := π; Fim-Se
11.         Se cnt >= n então
12.                 π := Perturbar(π*, pert);
13.                 Se f(π) < f(π*) então π* := π Fim-Se
14.                 cnt := 0;
15.         Fim – Se
16.     Fim – Para
17. Fim – Enquanto
18. Retorne π*
```

Figura 5 - Pseudocódigo ILS

```
Pertubar(π*, pert)
1.  π := π*;
2.  Para i := 1 até pert faça
3.      Selecione aleatoriamente duas tarefas adjacentes, π(i) e π(i+1) de π;
4.      π' := solução obtida pela troca π(i) e π(i+1);
5.  Fim – Para
6.  Return π
```

Figura 6 - Pseudocódigo da Perturbação

4.4 Algoritmo Genético

Nesta seção, são apresentados detalhes do algoritmo genético básico (AG-B) e do algoritmo genético com busca local (AG-BL). Na Figura 7 são representados os passos do AG-B; este algoritmo é composto primeiramente pela "geração da população inicial", a qual será "avaliada". As soluções de *Pop* são escolhidas aos pares aleatoriamente e as melhores são "selecionadas". O próximo passo é o "cruzamento" entre duas soluções e a "mutação", o que irá gerar uma nova população *Pop2*. Ao fim, *Pop2* sofre uma "avaliação" e o *Pop1* será atualizado. O AG-B é finalizado após atingir um critério de parada e a melhor solução será retornada.

```
AG básico(p_size, pc, pm, d, CondiçãoParada)
    1.  π := GeraPopulaçãoInicial(Pop);
    2.  π* := AvaliaPopulação(Pop);
    3.  Enquanto CondiçãoParada faça
    4.      S := Seleção(Pop);
    5.      Para i := 1 até p_size /2 faça
    6.          Selecione de S duas soluções pais: pai1 e pai2;
    7.              (filho1, filho2) = Crossover(pai1, pai2, pc);
    8.          filho1 := Mutação(filho1, pm);
    9.          filho2 := Mutação(filho2, pm);
    10.         Pop2 := Pop2 ∪ { filho1, filho2};
    11.     Fim-Para
    12.     π* := AvaliaPopulação (Pop2);
    13.     Pop := Pop2;
    14. Fim-Enquanto
Retorne π*
```

Figura 7 - Pseudocódigo do Algoritmo Genético Básico (AG-B)

Os passos do AG-BL são demonstrados na Figura 8. Além das etapas descritas no pseudocódigo do AG-B, existe a inclusão da busca local. A busca local é utilizada com o objetivo melhorar algumas soluções da população.

```
AG-BL(p_size, pc, pm, pbl, d, CondiçãoParada)
    1.  π := GeraPopulaçãoInicial(Pop);
    2.  π* := AvaliaPopulação(Pop);
    3.  Enquanto CondiçãoParada faça
    4.      S = Seleção(Pop);
    5.      Para i := 1 até p_size /2 faça
    6.          Selecione de S duas soluções pais: pai1 e pai2;
    7.              (filho1, filho2) = CrossoverSBOX(pai1, pai2, pc);
    8.          filho1 := Mutação(filho1, pm);
    9.          filho2 := Mutação(filho2, pm);
    10.         Pop2 := Pop2 ∪ { filho1, filho2};
    11.     Fim-Para
    12.     π* := AvaliaPopulação (Pop2);
    13.     Pop2 := BuscaLocal(Pop2, pbl, d, π*);
    14.     Pop := Pop2;
    15. Fim-Enquanto
Retorne π*
```

Figura 8 - Pseudocódigo do Algoritmo Genético com Busca Local (AG-BL)

Nas próximas subseções são descritas as etapas dos algoritmos: AG-B e AG-BL.

4.4.1 Geração da População Inicial

Neste trabalho, parte da população de soluções é gerada utilizando à heurística NEH, a qual utiliza 4 regras diferentes de despacho. Estas regras são as seguintes: SPT, EDD, MST e WSPT.

Utilizando a heurística NEH com uma das regras de despacho, são geradas $0{,}1\times p_{size}$ soluções, onde p_{size} é o tamanho da população. Ou seja, utilizando as quatro regras, 40% da população é gerada através da heurística NEH. As outras soluções da população são geradas de forma aleatória.

4.4.2 Seleção, Cruzamento e Mutação

O método de seleção utilizado no AG é a seleção por torneio (*tournament selection*) (MICHALEWICZ, 1996). Este método consiste em selecionar aleatoriamente duas soluções da população corrente *Pop* e escolhe-se a melhor, mais apta. Assim, p_{size} soluções são escolhidas e armazenadas em um conjunto *S*.

Na etapa de cruzamento, duas soluções, *pai1* e *pai2*, são escolhidas aleatoriamente do conjunto *S*. Estas soluções são cruzadas aplicando um operador de *crossover* e são gerados dois filhos, *filho1* e *filho2*. Neste trabalho foram testados dois operadores de *crossover*: o SBOX (*Similar Block Order Crossover*) e o SJOX (*Similar Job Order*) (RUIZ *et al*., 2006b). O *crossover* SBOX ilustrado na Figura 9 funciona da seguinte maneira: blocos de pelo menos duas tarefas presentes nos dois pais (nas mesmas posições) são copiados para os dois filhos. Já no *crossover* SJOX ilustrado na Figura 10, os dois filhos herdam as tarefas que estão nas mesmas posições, tanto no Pai 1 como no Pai 2, independentemente de bloco.

Em seguida, determina-se aleatoriamente um ponto de corte para dividir a sequência de tarefas. O *filho1* herda a primeira parte do *pai1* e a segunda parte é completada com as tarefas do *pai2* que ainda não foram inseridas nesse filho. Similarmente é construído o *filho2*. O operador de *crossover* foi testado com as probabilidades *pc* = 80% e 100%, sendo que no primeiro, algumas das soluções *pai1* e *pai2* não serão cruzados, elas ficarão no lugar dos filhos. Ao usar probabilidade *pc* = 100% todas as soluções *pai1* e *pai2* serão cruzadas.

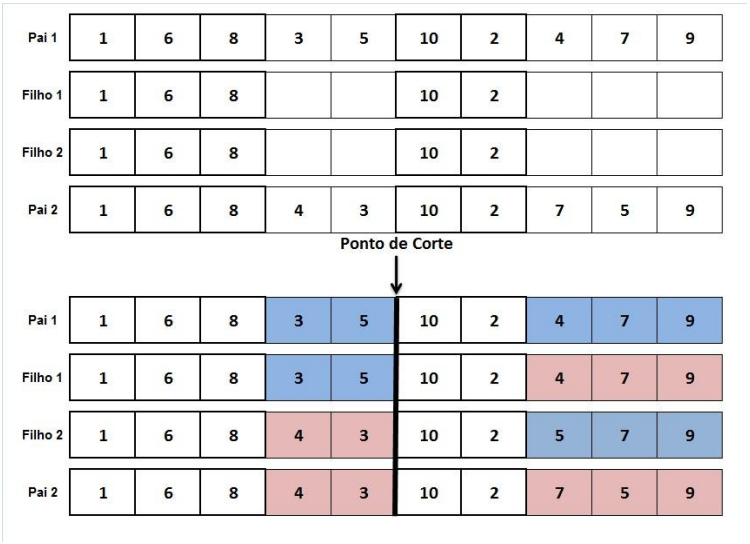

Figura 9 - Similar Block Order Crossover (SBOX)

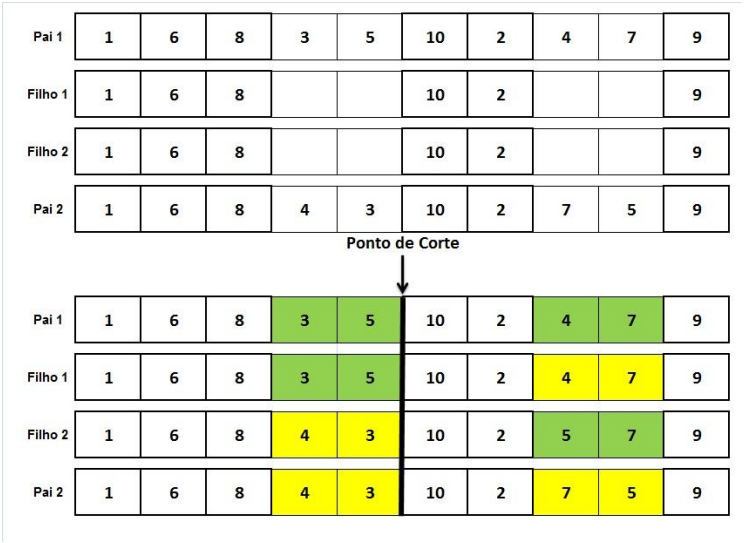

Figura 10 - Similar Job Order Crossover (SJOX)

O operador de mutação consiste em perturbar as soluções filhos (gerados pelo operador de *crossover*) da seguinte maneira: aleatoriamente, escolhe-se uma tarefa da sequencia para ser inserida em outra posição da sequência. Essa perturbação foi testada com a probabilidade p_m = 10% e 20%. A Figura 11 apresenta um exemplo do operador de mutação.

A cada iteração do AG, aplicam-se os operadores de *crossover* e mutação para gerar uma nova população *Pop2* de p_{size} soluções. As soluções desta população são avaliadas e a melhor solução π^* é atualizada. O AG finaliza após executar um número de iterações determinado por um critério de parada. A melhor solução encontrada é retornada.

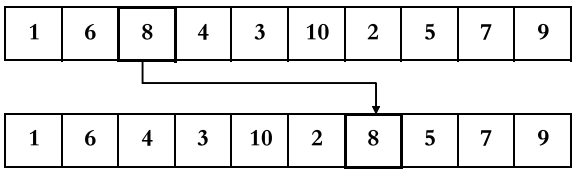

Figura 11 - Mutação por inserção

4.4.3 Busca Local

A busca local utilizada no AG-BL tem por objetivo melhorar algumas soluções geradas pelos operadores de *crossover* e de mutação (soluções da população *Pop2*). A probabilidade para que uma solução seja melhorada é *pbl*, a qual foi testada com 3% e 5%. Vale ressaltar que heurísticas de busca local baseadas em busca em vizinhança são demoradas; portanto, a probabilidade para que uma solução da população seja melhorada deve ser baixo.

O procedimento de busca local é composto de duas etapas: *destruição* e *reconstrução*. Na etapa de *destruição*, *d* tarefas são removidas da solução π a ser melhorada, obtendo uma solução parcial π_p. As tarefas removidas são armazenadas em π_R ($\pi_R(i)$, i=1,...,d, são as tarefas removidas). A etapa de *reconstrução* possui *d* passos. No passo i = 1, (n-d+1) soluções parciais são construídas inserindo-se a tarefa $\pi_R(1)$ em todas as possíveis posições de π_p. As (n-d+1) soluções parciais são avaliadas e a melhor solução é selecionada. Esta solução selecionada substitui a solução π_p. Os passos i=2,…,d são similares. Note que, no passo i = d, n soluções completas são geradas. Destas n soluções, seleciona-se a melhor que substituirá a solução π de *Pop2*. Neste trabalho foram testadas a remoção de d = 3, 6 e 8 tarefas. Na Figura 12 apresenta-se o pseudocódigo do procedimento de busca local.

BuscaLocal (*Pop2*, *pbl*, *d*)

1. **Para** cada solução π de *Pop2* **faça**
2. **Se** a probabilidade *pbl* é satisfeita **então**
3. $\pi_{p:}$= remover *d* tarefas de π; *(π_R = conjunto de tarefas removidas)*
4. **Para** *j* :=1,...,*d* **faça**
5. Inserir tarefa $\pi_R(j)$ em todas as posições de π_p, gerando *n-d+j* sequências parciais;
6. Avalie todas as sequências parciais geradas;
7. π := escolha a melhor sequência parcial;
8. **Fim-Para** (π_p será uma solução completa)
9. **Fim-Se**
10. **Se** $f(\pi_p) < f(\pi)$ **então**
11. $\pi := \pi_p$; *(a solução π é substituída)*
12. **Fim-Se**
13. **Se** $f(\pi_p) < f(\pi^*)$ **então**
14. $\pi^* := \pi_p$; *(a melhor solução é atualizada)*
15. **Fim-Se**
16. **Fim-Para**
17. **Retorne** *Pop2*

Figura 12 - Pseudocódigo da Busca Local

CAPÍTULO 5

EXPERIMENTOS COMPUTACIONAIS

Neste capítulo são apresentados os resultados das metaheurísticas propostas, detalhes e justificativas para utilização dos métodos. Na seção 5.1 é apresentado o ambiente computacional em que os testes foram realizados e a descrição do conjunto de instâncias do problema que foram utilizadas. Já na seção 5.2 são apresentados os parâmetros dos métodos propostos e a justificativa para sua utilização nos testes. E, finalmente, na seção 5.3, demonstram-se os resultados e comparações de desempenho entre os métodos heurísticos (NEH com diferentes regras de despacho) e as metaheurísticas.

São comparados os resultados das metaheurísticas propostas ILS, AG-B e AG-BL. As soluções geradas por estas metaheurísticas também são comparadas com as soluções determinadas por duas heurísticas construtivas, MST e NEH-MST. O objetivo é observar o ganho obtido pelas metaheurísticas com relação às heurísticas construtivas.

5.1 Ambiente de testes e instâncias utilizadas

Para realização dos testes, foram gerados um total de 310 instâncias do problema, de pequeno, médio e grande porte. Nas instâncias de pequeno porte, o número de tarefas $n \in \{8, 10, 12\}$, o número de estágios $k \in \{3, 4\}$ e o número m_t de máquinas em cada estágio t varia de 1 a 3 (isto é, m_t é gerado aleatoriamente no intervalo [1,3]). Nas instâncias do problema com n = 20, 30, 50, 80 tarefas, o número de estágios k varia no conjunto {5, 8, 10, 12, 15} e o número m_t de máquinas em cada estágio varia de 1 a 8. Nas instâncias do problema com n = 100 tarefas, o número de

estágios $k \in \{5, 10, 15, 20, 25\}$ e o número m_t de máquinas em cada estágio varia de 1 a 9. Para cada combinação de n e k, foram geradas 10 instâncias do problema.

Os tempos de processamento das tarefas foram gerados aleatoriamente no intervalo [1, 99]. Os tempos de preparação das máquinas de cada estágio são determinados aleatoriamente no intervalo [0, 9]. Vale ressaltar que, as máquinas em cada estágio possuem os mesmos *setups*. Por tanto é gerada, uma matriz de *setups* de ordem $n \times n$ para cada estágio, pois os *setups* são dependentes da sequência das tarefas. O *setup* das máquinas, antes de processar a primeira tarefa é 0 (zero).

As datas de entrega são geradas de acordo com a proposta de Hasija e Rajendran (2004). Para geração das datas de entrega aplicam-se os seguintes passos:

1) Para cada tarefa j calcula-se o tempo total de processamento em todos os estágios:

$$\forall j = 1,...n,\ P_j = \sum_{t=1}^{k} P_{jt} \tag{20}$$

2) Para cada tarefa j, calcula-se a soma dos setups médios em cada estágio t:

$$\forall j = 1,...n,\ S_j = \frac{\sum_{t=1}^{k} \sum_{i=1, i \neq j}^{n} S_{tji}}{n-1} \tag{21}$$

3) Para cada tarefa j, calcula-se a data de entrega d_j da seguinte maneira:
   $d_j = (1+3r) \times (P_j + S_j)$, onde r é um número real gerado aleatoriamente no intervalo [0,1].

Com esta forma de geração de datas de entrega, são geradas datas muito apertadas a relativamente apertadas que dependem do valor do número r. Se o número r é próximo de 0, então a data de entrega da tarefa é muito apertada e esta data seria aproximadamente $(P_j + S_j)$. Se o número r é próximo de 1, então a data de entrega é aproximadamente $4 \times (P_j + S_j)$.

As penalidades por atraso e de adiantamento foram geradas aleatoriamente no intervalo [1, 9] e o instante de liberação das tarefas são geradas aleatoriamente no intervalo [1, 10].

As heurísticas foram implementadas na linguagem de programação C++, utilizou-se o compilador Microsoft Visual C++ 2010 Express e executadas em um computador Intel Core i5, com 2,67 GHz e 6 GB de memória, sob plataforma Windows 7.

Soluções de problemas de pequeno porte são obtidas através da resolução do modelo matemático de PLI (apresentado na seção 3). Para isto, é utilizado o software de otimização da IBM, ILOG CPLEX versão 12.2.

5.2 Metodologia para avaliação dos métodos propostos

Os métodos heurísticos implementados neste trabalho utilizam alguns parâmetros, que devem ser determinados experimentalmente, antes da execução das instâncias de teste.

Os resultados obtidos foram analisados e comparados, utilizando a média do desvio relativo percentual - RPD (*Relative Percentual Deviation)*. Esta medida de desempenho é calculada conforme Vallada e Ruiz, (2009), através da seguinte fórmula:

$$RPD\ \% = 100 \times \frac{Metodo_{sol} - Melhor_{sol}}{Melhor_{sol}}, \qquad (22)$$

onde, $Metodo_{sol}$ é a média do resultado (solução) obtido através de um dos métodos e $Melhor_{sol}$ é o melhor resultado obtido para um problema, dentre todas as execuções e todas as versões do algoritmo. Note que o RPD mede o desvio de uma solução, em relação à melhor solução. Para as heurísticas NEH-MST, ILS, AG-B e AG-BL, o RPD foi calculado utilizando o resultado médio de 5 execuções.

A fim de validar alguns dos resultados apresentados, foi realizada uma análise de variância (ANOVA)(MONTGOMERY, 2007), para determinar se existe uma diferença do RPD médio que seja estatisticamente significante. Para esta comparação utilizou-se o teste estatístico de TUKEY (p<0,05), onde p representa a probabilidade de confiança.

5.3 Análises dos parâmetros usados nos métodos heurísticos

Nesta seção, inicialmente são analisados os parâmetros da heurística NEH, em seguida as heurísticas construtivas para escolha das melhores a serem utilizadas, é apresentado também o critério para a condição de parada e os parâmetros das metaheurísticas: ILS, AG-B e AG-BL.

5.3.1 Heurísticas construtivas

No capítulo 4 deste trabalho foi apresentada a utilização da heurística NEH, nas metaheurísticas ILS, AG-B e AG-BL. A heurística NEH possui um parâmetro, que

o é valor da taxa de aleatoriedade (α). Os testes foram realizados na tentativa de definir um valor para α que proporcione bons resultados.

Para analisar os valores de α foram testadas as regras: NEH-SPT, NEH-LPT, NEH-MST, NEH-EDD, NEH-ERD, NEH-WSPT e NEH-WLPT. Utilizou-se a medida de desempenho RPD e 90 instâncias do problema compostas por n = 20, 50, 100 tarefas, o número de estágios k varia no conjunto {5, 10, 20} e o número m_t de máquinas em cada estágio varia de 1 a 9. Cada regra foi executada com as 90 instâncias do problema e testada nos intervalos de α, conforme descrito:

α = 0,0 (para $n \leq 20$), α = 0,0 (para $n \geq 30$ & $n \leq 50$) e α = 0,0 (para $n \geq 80$ & n ≤ 100);

α = 0,6 (para $n \leq 20$), α = 0,4 (para $n \geq 30$ & $n \leq 50$) e α = 0,2 (para $n \geq 80$ & n ≤ 100);

α = 0,5 (para $n \leq 20$), α = 0,3 (para $n \geq 30$ & $n \leq 50$) e α = 0,2 (para $n \geq 80$ & n ≤ 100);

α = 0,4 (para $n \leq 20$), α = 0,2 (para $n \geq 30$ & $n \leq 50$) e α = 0,1 (para $n \geq 80$ & n ≤ 100).

Na Figura 13 ilustra-se o RPD médio entre as regras NEH-SPT, NEH-LPT, NEH-MST, NEH-EDD, NEH-ERD, NEH-WSPT e NEH-WLPT, nos quatro intervalos de α. Nesta figura observa-se que o intervalo de α = 0,6 (para $n \leq 20$), α = 0,4 (para n $\geq$ 30 & n $\leq$ 50) e α = 0,2 (para n $\geq$ 80 & n $\leq$ 100), apresenta uma menor dispersão média (variação ou variabilidade).

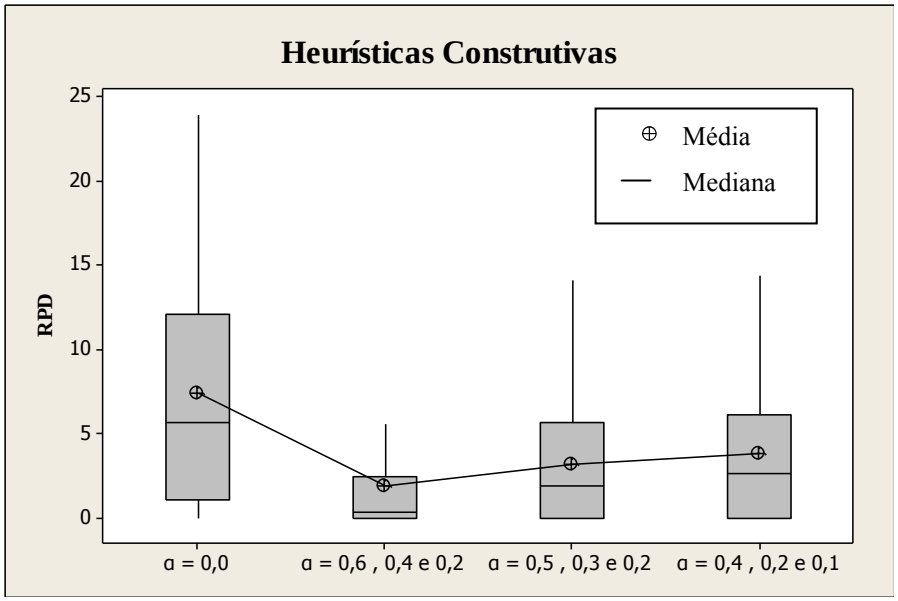

Figura 13 - Boxplot das taxas de aleatoriedade (α) da heurística NEH

Na Tabela 3 são apresentadas as médias RPD e as medianas, referentes ao gráfico da Figura 13. A menor média de RPD é 1,94, referente aos valores de α = 0,6, α = 0,4 e α = 0,2.

Tabela 3 - RPDs médio e mediana das taxas de aleatoriedade (*α*) da heurística NEH

Taxas de Aleatoriedade (*α*)	Média	Mediana
$\alpha = 0,0$	7,43	5,69
$\alpha = 0,6$, $\alpha = 0,4$ e $\alpha = 0,2$	**1,94**	0,38
$\alpha = 0,5$, $\alpha = 0,3$ e $\alpha = 0,2$	3,28	1,96
$\alpha = 0,4$, $\alpha = 0,2$ ε $\alpha = 0,1$	3,87	6,17

Para validar os resultados dos intervalos de α é interessante verificar se as diferenças entre as médias de RPD são estatisticamente significantes. Os testes estatísticos foram feitos utilizando o *software* estatístico Minitab na versão 14, com um nível de significância de 95% que determina a confiabilidade do teste. Foi realizada uma análise de variância seguindo o procedimento ANOVA, para testar a significância estatística das diferenças entre as médias do RPD através de comprovação ou rejeição de uma hipótese nula. O resultado do teste ANOVA é apresentado na Figura 14. As colunas DF, **F** e P representam respectivamente o grau de liberdade, o valor de **F** do ANOVA e a probabilidade de aceitação da hipótese nula. Foi utilizado o teste de TUKEY (p<0,05), onde p representa a probabilidade de confiança. Analisando o resultado é possível verificar que $p = 0$ e que nenhum intervalo probabilístico passa pelo ponto zero. Este resultado do teste ANOVA, comprova que a diferença entre o RPD médio é estatisticamente significante. Outra observação é que o intervalo de confiança individual é de 98,93%.

```
Source   DF        F      P
Factor    3    20,48  0,000
Error   356
Total   359

                                    Individual 95% CIs For Mean Based on
                                    Pooled StDev
Level               Mean    -----+---------+---------+---------+----
α  = 0,0            7,428                              (----*----)
α  = 0,6 , 0,4 e    1,937    (----*----)
α  = 0,5 , 0,3 e    3,276          (----*----)
α  = 0,4 , 0,2 e    3,866             (----*----)
                              -----+---------+---------+---------+----
                                 2,0       4,0       6,0       8,0


Tukey 95% Simultaneous Confidence Intervals
All Pairwise Comparisons

Individual confidence level = 98,93%
```

Figura 14 - Teste de comparação múltipla Tukey do parâmetro *α*

Foram testadas diferentes regras de despacho para escolha das melhores a serem utilizadas com as metaheurísticas: ILS, AG-B e AG-BL. Entre elas, estão às

regras WSPT, MST, EDD, WLPT, LPT, SPT, ERD e a adaptação destas com a regra NEH.

Para a realização deste teste, utilizou-se o RPD médio e grupos compostos por n = 20, 50, 100 tarefas, o número de estágios k variando no conjunto de {5, 10, 20} e o número m_t de máquinas variando de 1 a 9 em cada estágio. Cada grupo foi composto por 10 instâncias. Para as heurísticas NEH foram utilizados os valores de $\alpha = 0{,}6$ (para $n \leq 20$), $\alpha = 0{,}4$ (para $n \geq 30$ & $n \leq 50$) e $\alpha = 0{,}2$ (para $n \geq 80$ & $n \leq 100$).

Na Figura 15, estão apresentadas as médias RPD das heurísticas construtivas. Observa-se que as regras: NEH-MST, NEH-EDD, NEH-WSPT e NEH-SPT apresentaram melhores resultados.

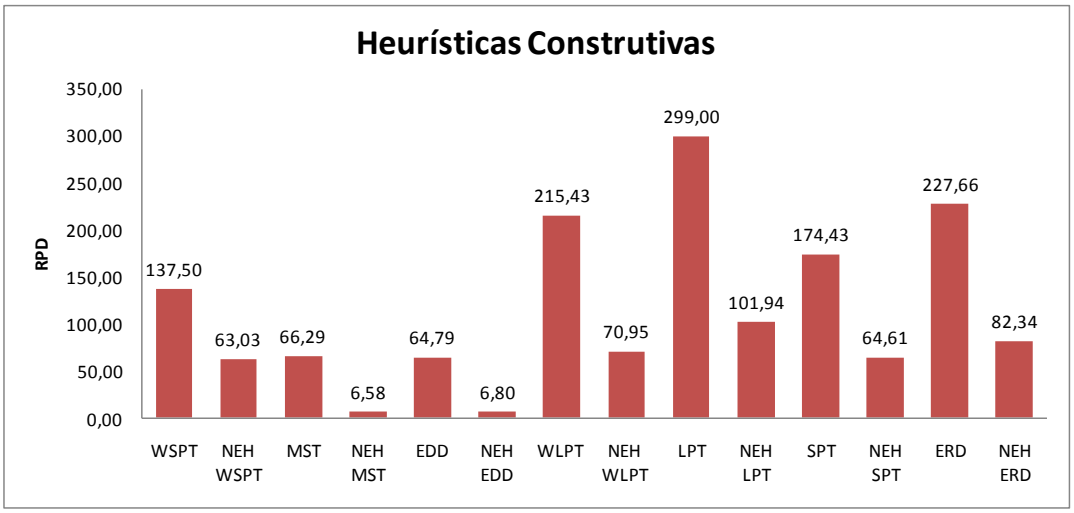

Figura 15 – Comparação da heurística NEH com as diferentes regras de despacho

Entre as regras testadas, as que utilizam datas de entrega apresentam melhores resultados para o problema descrito neste trabalho: MST e EDD, sendo que estas ainda podem ser melhoradas através da combinação com a heurística NEH.

5.3.2 Condição de parada das metaheurísticas

Um parâmetro comum que as metaheurísticas desenvolvidas (ILS, AG-B e AG-BL) possuem é o critério de parada (*CondiçãoParada*). Para este critério geralmente é utilizado um número fixo de iterações.

O objetivo deste trabalho é testar o desempenho destas metaheurísticas. Para isso, utiliza-se uma mesma condição de parada que é baseada em tempo de CPU. Este

tempo deve ser dependente do tamanho do problema, ou seja, número de tarefas *n* e número de estágios *k*. Vale ressaltar que o espaço de soluções do problema de *flowshop* cresce exponencialmente de acordo ao número de tarefas.

Para determinar o tempo máximo de execução dos algoritmos foi definida a seguinte expressão (este tempo depende principalmente do número de tarefas):

$$TempoMax = (n^2 \times \sqrt{k}) \times 15 \text{ milissegundos} \quad (23)$$

Os tempos de CPU (em segundos) utilizados pelos algoritmos são apresentados na Tabela 4. Note que os tempos apresentam um crescimento gradativo de acordo com o número de tarefas *n* e estágios *k*.

Tabela 4 - Tempo de CPU (em segundos)

n	*k*	tempos (s)
8	3	1,66
8	4	1,92
10	3	2,60
10	4	3,00
12	3	3,74
12	4	4,32
20	5	13,42
20	8	16,97
20	10	18,97
20	12	20,78
20	15	23,24
30	5	30,19
30	8	38,18
30	10	42,69
30	12	46,77
30	15	52,29
50	5	83,85
50	8	106,07
50	10	118,59
50	12	129,90
50	15	145,24
80	5	214,66
80	8	271,53
80	10	303,58
80	12	332,55
80	15	371,81
100	5	335,41
100	10	474,34
100	15	580,95
100	20	670,82
100	25	750,00

5.3.3 Parâmetro de entrada da metaheurística ILS

A metaheurística ILS possui um parâmetro de entrada *pert* (número de trocas) utilizado na etapa de perturbação. Para este parâmetro foram testados seis valores

diferentes, *pert* = 4, 6, 8, 10, 12 e 15. Utilizando cada um destes valores, o algoritmo ILS foi executado para resolver 210 instâncias do problema, sendo cada um resolvido 5 vezes usando diferentes sementes. Como solução inicial para este teste, foi utilizada a heurística construtiva NEH MST.

Na Tabela 5 é apresentado o resultado de *pert*, através da média RPD. Observa-se que, quando se aumenta o valor de *pert*, a variação entre os resultados das médias RPD é muito pequena. Desta forma, foi escolhido o parâmetro *pert* = 15, para execução da metaheurística ILS. Como a diferença entre os dois últimos resultados *pert* = 12 e *pert* = 15 foi pequena, outros valores para *pert* > 15 não foram testados.

Tabela 5 - Médias RPD da metaheurística ILS para os diferentes valores de *pert*

Problema	***n***	***k***	**RPD médio**					
			***pert* = 4**	***pert* = 6**	***pert* = 8**	***pert* = 10**	***pert* = 12**	***pert* = 15**
G1	8	3	0,000	0,000	0,000	0,000	0,000	0,000
G2	8	4	0,000	0,000	0,000	0,000	0,000	0,000
G3	10	3	0,000	0,000	0,000	0,000	0,000	0,000
G4	10	4	0,000	0,000	0,000	0,000	0,000	0,000
G5	12	3	0,000	0,000	0,000	0,000	0,000	0,000
G6	12	4	0,000	0,000	0,000	0,000	0,000	0,000
G7	20	5	**0,000**	0,001	0,001	0,007	0,007	0,001
G8	20	8	0,016	0,020	0,024	0,013	**0,008**	0,014
G9	20	10	0,000	0,000	0,000	0,000	0,000	0,002
G10	20	12	0,009	0,009	0,012	0,016	0,005	0,005
G11	20	15	**0,002**	0,004	0,011	0,005	0,003	0,009
G12	30	5	0,210	0,162	0,156	**0,132**	0,143	0,160
G13	30	10	**0,188**	0,469	0,417	0,282	0,239	0,248
G14	30	15	0,029	0,028	0,023	0,029	**0,015**	0,021
G15	50	5	0,910	0,944	0,952	0,984	**0,654**	0,884
G16	50	10	0,745	0,750	0,630	0,607	0,760	**0,537**
G17	50	15	0,455	0,440	0,398	0,447	**0,363**	0,480
G18	80	8	0,761	0,719	**0,624**	0,800	0,795	0,813
G19	80	12	1,759	1,590	1,534	1,295	1,620	**1,262**
G20	100	10	1,307	**0,988**	1,113	1,208	1,193	1,283
G21	100	20	2,246	1,944	1,968	1,979	1,627	**1,564**

Para gerar uma solução inicial para o algoritmo ILS, foram testadas as heurísticas NEH-MST e NEH-EDD. Estas heurísticas apresentaram os melhores resultados (veja os testes das heurísticas construtivas na seção 5.3.1).

Na Tabela 6 são apresentadas as médias dos RPDs das 5 execuções do ILS com NEH-MST e ILS com NEH-EDD. Para cada combinação de *n* e *k,* foram testados 10 problemas.

É possível constatar que na maioria das médias RPD, referentes à execução das 310 instâncias do problema, a metaheurística ILS utilizando a heurística construtiva NEH-MST apresenta melhores resultados.

Tabela 6 - Heurísticas construtivas NEH-MST e NEH-EDD para o ILS

Problema	n	k	ILS	
			NEH-MST	NEH-EDD
G1	8	3	0,000	0,000
G2	8	4	0,000	0,000
G3	10	3	0,000	0,000
G4	10	4	0,000	0,000
G5	12	3	0,000	0,000
G6	12	4	0,000	0,000
G7	20	5	**0,001**	0,004
G8	20	8	**0,014**	0,027
G9	20	10	0,002	**0,000**
G10	20	12	**0,005**	0,039
G11	20	15	**0,009**	0,013
G12	30	5	**0,160**	0,238
G13	30	8	**0,058**	0,137
G14	30	10	**0,248**	0,411
G15	30	12	**0,005**	0,027
G16	30	15	**0,017**	0,031
G17	50	5	**0,717**	0,807
G18	50	8	**0,299**	0,829
G19	50	10	**0,391**	0,865
G20	50	12	**0,658**	1,003
G21	50	15	0,403	**0,341**
G22	80	5	**0,313**	0,507
G23	80	8	**0,408**	0,905
G24	80	10	1,634	**1,429**
G25	80	12	**1,103**	1,136
G26	80	15	**1,000**	1,510
G27	100	5	**1,648**	1,988
G28	100	10	**0,724**	0,899
G29	100	15	**0,946**	1,558
G30	100	20	**1,015**	2,582
G31	100	25	**0,705**	2,092
	Média		**0,403**	0,625

Na Figura 16 apresenta-se um *boxplot* da distribuição das médias de RPD nos testes utilizando ILS com as heurísticas construtivas: NEH-MST e NEH-EDD. Nesta figura, observa-se que o ILS com NEH-MST obteve uma menor dispersão média. Na Tabela 6 são descritas médias de RPD e medianas referentes ao gráfico da Figura 16. Observa-se que o algoritmo ILS, utilizando NEH-MST, obteve melhor resultado.

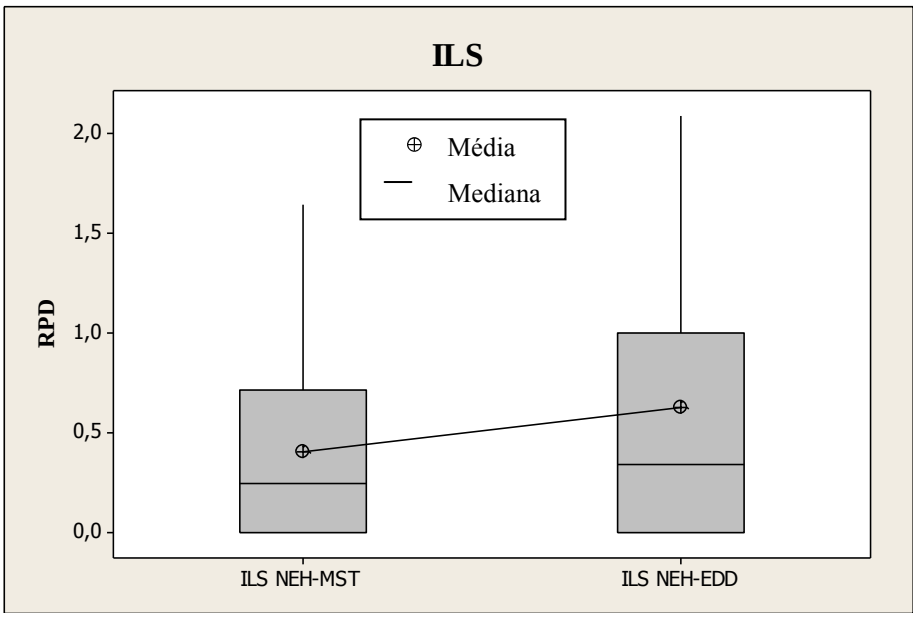

Figura 16 - Boxplot da comparação das heurísticas construtivas para o ILS

Tabela 7 - RPDs médio e medianas do ILS com as heurísticas construtivas

ILS	*Média*	*Mediana*
NEH-MST	**0,4027**	0,248
NEH-EDD	0,6251	0,341

5.3.4 Análise dos parâmetros do Algoritmo Genético Básico (AG-B)

Para o AG-B foram analisados os parâmetros probabilidade de mutação *pm* e probabilidade de cruzamento *pc* e foram testados dois tipos de cruzamento (operador de *crossover*)

Na tabela 8 são apresentadas as combinações testadas para o AG-B, sendo 80% e 100% de *pc*, 10% e 20% de *pm* e dois operadores de *crossover* SBOX e SJOX.

Tabela 8 - Combinações entre *pm* e *pc* e os tipos de cruzamento do AG-B

pm	*pc*	*crossover*
10%	80%	SBOX
10%	80%	SJOX
10%	100%	SBOX
10%	100%	SJOX
20%	80%	SBOX
20%	80%	SJOX
20%	100%	SBOX
20%	100%	SJOX

Através da análise dos resultados apresentados na Tabela 9, é possível constatar que o *crossover* SJOX, apresenta melhores resultados em todos os casos, ao contrário do SBOX. A combinação de parâmetros com melhores resultados em relação à média RPD foi *pm* = 20%, *pc* = 80 e *crossover* = SJOX.

Tabela 9 - RPDs médio das combinações entre *pm* e *pc* e os tipos de cruzamento do AG-B

Problema	*n*	*k*	*pm=10% pc=80 crossoverSBOX*	*pm=10% pc=80 crossover SJOX*	*pm=10% pc=100 crossover SBOX*	*pm=10% pc=100 crossover SJOX*	*pm=20% pc=80 crossover SBOX*	*pm=20% pc = 80 crossover SJOX*	*pm=20% pc=100 crossover SBOX*	*pm=20% pc=100 crossover SJOX*
G1	8	3	1,17	0,11	0,80	0,00	0,93	0,00	0,65	0,00
G2	8	4	1,08	**0,15**	1,24	0,81	0,33	0,16	0,50	0,16
G3	10	3	7,70	0,77	9,42	0,73	4,84	0,53	6,67	**0,51**
G4	10	4	39,59	36,67	38,34	34,35	37,31	32,84	37,96	**33,33**
G5	12	3	11,16	0,20	10,67	0,06	10,08	0,00	10,24	0,00
G6	12	4	11,60	1,30	11,03	0,91	9,27	**0,33**	9,66	0,52
G7	20	5	40,91	7,02	39,57	**7,00**	38,20	7,43	39,15	7,64
G8	20	8	25,97	5,86	26,29	6,03	26,92	**4,93**	26,63	5,71
G9	20	10	20,80	3,98	21,38	4,55	20,71	3,56	21,30	**3,48**
G10	20	12	14,11	3,58	14,28	4,10	13,49	**2,89**	13,97	3,36
G11	20	15	17,82	8,89	17,89	9,40	18,29	**8,59**	17,21	9,79
G12	30	5	45,96	**7,02**	45,69	7,04	47,11	4,80	46,85	5,97
G13	30	10	46,42	27,43	46,57	3,33	46,77	**1,87**	47,56	2,79
G14	30	15	24,13	1,47	24,57	2,05	24,42	**1,33**	24,84	1,39
G15	50	5	69,90	5,17	68,30	5,87	69,36	4,84	70,43	**4,49**
G16	50	10	71,13	5,48	70,72	6,43	71,61	**4,29**	71,52	5,22
G17	50	15	85,30	3,70	84,18	3,86	85,46	**2,74**	84,60	3,24
G18	80	8	100,65	**2,77**	100,35	3,42	101,37	3,61	99,91	3,92
G19	80	12	113,90	5,92	112,34	4,73	113,68	**4,66**	113,34	5,61
G20	100	10	99,25	**3,62**	99,17	4,16	99,24	4,05	98,91	4,21
G21	100	20	122,56	5,06	122,96	**4,82**	123,29	4,93	121,97	4,77
Média			46,24	6,48	45,99	5,41	45,84	**4,69**	45,90	5,05

5.3.5 Parâmetros do Algoritmo Genético com Busca Local (AG-BL)

Para o AG-BL foi usada a melhor combinação de *pm* e *pc* e do operador de *crossover*. Nesta seção, testam-se diferentes valores para o parâmetro *d* usado na busca local e a probabilidade de busca local (*pbl*)

A busca local é utilizada com o objetivo de melhorar alguns resultados gerados pelos operadores de *crossover* e de mutação. Neste trabalho, as *pbl* testadas foram de 3% e 5%. Já o parâmetro *d*, refere-se à etapa de *destruição*, onde tarefas são removidas

da solução ser melhorada, obtendo uma solução parcial. Para este parâmetro *d* os testes realizados, foram com 3, 6 e 8 tarefas. Na Tabela 10, é apresentada a combinação dos parâmetros utilizados no AG-BL.

Tabela 10 - Combinações dos parâmetros do AG-BL

pbl	*d*
3%	3
3%	6
3%	8
5%	3
5%	6
5%	8

Na Tabela 11, são demonstrados os resultados do RPD médio dos testes realizados com as combinações dos parâmetros do AG-BL. Os valores dos parâmetros que obtiveram melhores resultados foram: *pbl* = 5% e *d* = 6.

Tabela 11- Médias do RPD dos parâmetros do AG-BL

Problema	***n***	***k***	***pbl* = 3% *d* = 3**	***pbl* = 3% *d* = 6**	***pbl* = 3% *d* = 8**	***pbl* = 5% *d* = 3**	***pbl* = 5% *d* = 6**	***pbl* = 5% *d* = 8**
G1	8	3	0,000	0,000	0,000	0,000	0,000	0,000
G2	8	4	0,000	0,000	0,000	0,000	0,000	0,000
G3	10	3	0,000	0,000	0,000	0,000	0,000	0,000
G4	10	4	1,428	1,258	2,101	1,713	1,970	1,141
G5	12	3	0,000	0,000	0,000	0,000	0,000	0,000
G6	12	4	0,000	0,000	0,000	0,000	0,000	0,000
G7	20	5	8,746	9,710	10,224	10,310	9,155	9,222
G8	20	8	8,839	9,103	8,408	9,086	7,713	7,904
G9	20	10	4,147	4,818	3,387	3,292	4,174	2,529
G10	20	12	2,137	2,220	2,187	2,111	2,372	2,253
G11	20	15	9,264	8,435	8,271	7,941	8,531	9,119
G12	30	5	3,881	4,688	5,021	4,974	4,397	5,078
G13	30	10	2,826	3,591	3,594	3,524	3,103	3,593
G14	30	15	1,182	0,821	0,799	1,096	0,971	1,187
G15	50	5	6,046	7,360	7,768	5,933	5,737	6,442
G16	50	10	3,604	3,161	4,981	2,698	2,574	4,261
G17	50	15	2,589	2,516	3,079	2,051	1,861	2,924
G18	80	8	2,385	2,454	2,519	1,990	1,648	2,600
G19	80	12	4,353	4,719	4,969	2,611	3,045	4,940
G20	100	10	3,232	3,451	3,762	2,827	2,148	4,090
G21	100	20	5,446	5,270	6,466	2,624	3,526	6,122
Média			3,338	3,504	3,692	3,085	**2,996**	3,496

5.4 Resultados obtidos e comparações

Nesta seção são comparadas as médias do RPD das heurísticas construtivas MST e NEH-MST, das metaheurísticas propostas ILS, AG-B e AG-BL e do *software* CPLEX (para problemas de pequeno porte). Devido à complexidade do modelo matemático, a execução do CPLEX foi limitada em tempo de CPU. Para resolver os problemas com 8, 10 e 12 tarefas, o CPLEX é executado no máximo duas horas (7200 segundos). Vale ressaltar que, para alguns problemas, o CPLEX não consegue encontrar a solução ótima nesse tempo. A solução obtida é o melhor *upper bound*.

Para cada problema, os algoritmos ILS, AG-B e AG-BL foram executados 5 vezes. Os resultados médios de RPD (das 5 execuções) dos algoritmos são comparados. Estas médias também são comparadas com o RPD médio das heurísticas construtivas MST e NEH-MST e com os *upper bounds* determinados pelo CPLEX (para problemas de pequeno porte).

Todos os métodos são comparados utilizando o desvio relativo percentual (RPD) definido na Equação (24). Ressalta-se que, para cada problema, determina-se o melhor resultado ($Melhor_{sol}$), obtido por todos os métodos. Nessa equação, ($Melhor_{sol}$) é o valor da melhor solução encontrada entre todos os métodos e $Metodo_{sol}$ é o valor da solução determinada por um método, sendo que para as metaheurísticas ILS, AG-B e AG-BL, $Metodo_{sol}$ é o valor médio das 5 execuções realizadas.

A Tabela 12 apresenta a comparação entre os métodos CPLEX, MST, NEH-MST, ILS, AG-B e AG-BL. Nesta tabela, são apresentados os grupos de problemas, os parâmetros que determinam o tamanho destes problemas, número de tarefas n e número de estágios k, para cada (n|k) existem 10 problemas. Na outras colunas da Tabela 12 são mostradas as médias do RPD para cada método.

Tabela 12 - Comparação do Cplex e os Métodos Heurísticos (RPD médio)

Problemas	*n*	*k*	Cplex	MST	NEH-MST	AG-B	AG-BL	ILS
G1	8	3	**0.26**	117.01	44.70	13.74	13.74	13.74
G2	8	4	**0.86**	120.49	34.67	1.72	1.55	1.55
G3	10	3	4.84	143.76	35.55	2.89	2.37	2.37
G4	10	4	11.08	93.92	23.99	6.52	**6.74**	9.47
G5	12	3	5.02	97.66	15.71	0.70	0.70	0.70
G6	12	4	18.53	105.05	24.64	0.33	0.00	0.00
G7	20	5	-	83.37	28.96	7.85	**5.52**	7.65
G8	20	8	-	58.80	24.00	4.06	**2.55**	4.68
G9	20	10	-	57.03	22.29	3.50	3.65	**4.78**
G10	20	12	-	25.28	12.90	1.13	1.40	1.63
G11	20	15	-	44.07	23.22	7.56	**6.81**	12.77
G12	30	5	-	100.84	33.65	5.09	2.74	**2.19**
G13	30	8	-	86.42	28.59	6.74	**4.37**	5.93
G14	30	10	-	72.92	30.36	3.53	2.39	**2.03**
G15	30	12	-	72.76	25.97	3.70	**2.45**	2.80
G16	30	15	-	44.50	19.16	1.64	0.97	**0.99**
G17	50	5	-	133.99	39.52	10.01	4.80	**2.33**
G18	50	8	-	126.45	38.85	7.20	3.19	**0.62**
G19	50	10	-	168.85	37.62	6.54	2.39	**0.59**
G20	50	12	-	137.36	37.92	6.68	3.84	**0.98**
G21	50	15	-	102.36	37.14	4.22	1.77	**1.07**
G22	80	5	-	131.76	26.48	4.07	2.11	**0.27**
G23	80	8	-	157.34	51.19	7.03	2.17	**0.40**
G24	80	10	-	144.38	37.89	9.17	5.25	**1.49**
G25	80	12	-	164.09	52.13	8.52	4.23	**0.88**
G26	80	15	-	196.78	53.85	9.51	5.43	**0.91**
G27	100	5	-	142.17	34.24	11.47	5.48	**1.65**
G28	100	10	-	148.06	34.76	7.53	2.80	**0.66**
G29	100	15	-	161.05	47.14	7.95	6.07	**1.13**
G30	100	20	-	184.15	43.93	8.86	5.66	**1.14**
G31	100	25	-	138.24	44.53	8.41	7.81	**0.70**
Média				114.87	33.73	6.06	3.90	**2.84**

Na Tabela 12, observa-se que o CPLEX obteve as melhores soluções para os grupos de problemas G1 e G2. Para os grupos G3 a G6, o CPLEX determina apenas *upper bounds* bastante distantes da solução ótima. Na tabela, também pode ser observado que para os grupos de problemas G4, G7, G8, G11, G13, e G15, o AG-BL apresenta melhores resultados considerando a média do RPD. Para os problemas dos grupos G9, G12, G14, G16 a G31, a metaheurística ILS é a melhor com relação aos demais métodos apresentados.

Na Figura 17, ilustram-se os resultados do RPD médio para as metaheurísticas: ILS, AG-B e AG-BL. Observa-se que nos grupos de G1 até G16, existe muita

competitividade entre os resultados da média RPD nas metaheurísticas ILS, AG-B e AG-BL, nos grupos de G17 até G31 ILS é bastante superior aos outros métodos.

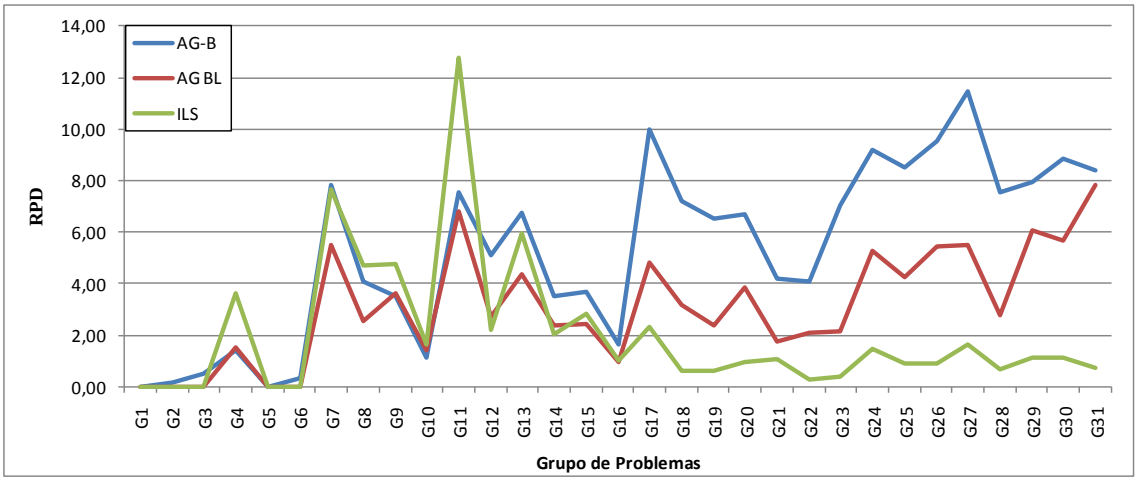

Figura 17 - Comparação entre as metaheurísticas

Na Figura 18 são ilustradas, graficamente, as médias de RPD e medianas das metaheurísticas: ILS, AG-B e AG-BL e os valores são apresentados na Tabela 13. É possível observar que a metaheurística ILS possui melhor média RPD em relação às demais.

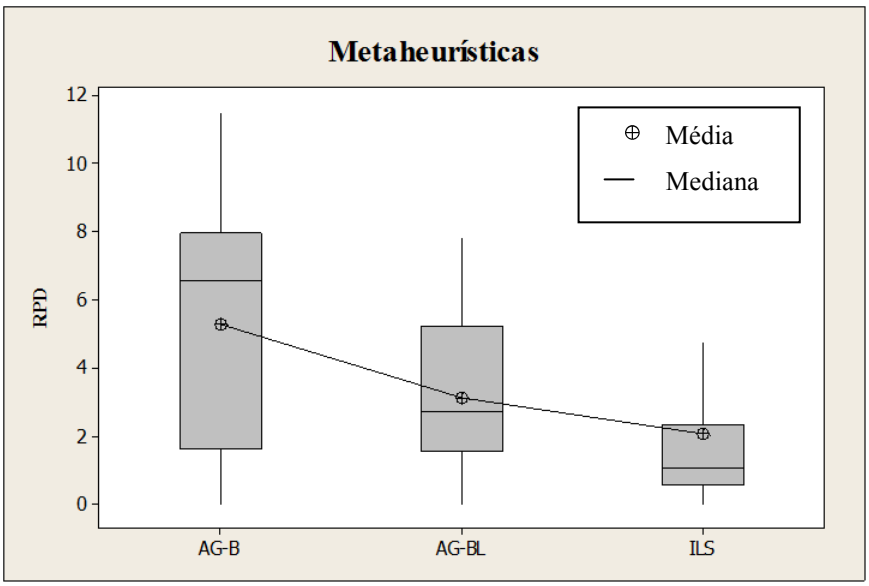

Figura 18 - Boxplot da comparação das metaheurísticas

Tabela 13 – RPDs médio e medianas das metaheurísticas

Metaheurísticas	*Média*	*Mediana*
AG-B	5,30	6,54
AG-BL	3,14	2,74
ILS	**2,02**	1,07

A fim de, validar os resultados do RPD médio das metaheurísticas, foi realizada a análise de variância seguindo o procedimento ANOVA. O resultado do teste ANOVA é apresentado na Figura 19. Vale ressaltar que, as colunas DF, **F** e P representam respectivamente o grau de liberdade, o valor de **F** do ANOVA e a probabilidade de aceitação da hipótese nula. Na Figura 19 apresenta teste de TUKEY com p = 0 e é possível observar ainda que nenhum intervalo probabilístico passa pelo ponto zero. Este resultado do teste ANOVA, comprova que a diferença entre o RPD médio é estatisticamente significante. Outra observação é que o intervalo de confiança individual é de 98,07%.

```
Source  DF   F      P
Factor   2  10,63  0,000
Error   90
Total   92

                        Individual 95% CIs For Mean Based on
                        Pooled StDev
Level  Mean    ---+---------+---------+---------+------
AG-B   5,303                        (-----*------)
AG-BL  3,142           (------*------)
ILS    2,061    (------*-----)
               ---+---------+---------+---------+------
                1,5       3,0       4,5       6,0

Tukey 95% Simultaneous Confidence Intervals
All Pairwise Comparisons

Individual confidence level = 98,07%
```

Figura 19 - Teste de comparação múltipla Tukey das metaheurísticas

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi abordado o problema de programação de tarefas em um ambiente *flowshop* flexível, tendo como objetivo a minimização das penalidades por adiantamentos e atrasos. Inicialmente, foi desenvolvido um modelo matemático para o problema. Para problemas de pequeno porte (com 8, 10 e 12 tarefas), o modelo foi resolvido utilizando o software de otimização CPLEX. Devido à complexidade do problema, o software não foi capaz de obter a solução ótima, em um tempo de duas horas.

Os parâmetros dos métodos utilizados foram descritos no capítulo 5, juntamente com os resultados obtidos nos testes computacionais. Através dos resultados, é possível verificar que os objetivos deste trabalho foram alcançados, ou seja, com uso de metaheurísticas que utilizam heurísticas construtivas, foi possível obter melhora de qualidade das soluções geradas. Outro ponto que deve ser abordado, é que as metaheurísticas utilizam tempo computacional relativamente baixo.

Nas instâncias de pequeno porte (com 8, 10 e 12 tarefas), o software de CPLEX, foi executado durante 2 horas, obtendo melhor resultado somente nas instâncias com 8 tarefas. Para as demais instâncias, os métodos propostos obtiveram melhores resultados.

Com o objetivo de obter soluções aproximadas foram desenvolvidas metaheuristicas de busca: ILS, AG-B e AG-BL. Para demonstrar o ganho das metaheurísticas em relação a heurísticas simples, foram implementadas duas heurísticas construtivas, MST e NEH-MST. Os métodos MST, NEH-MST, ILS, AG-B e AG-BL foram testados em um conjunto de 310 problemas. Dos resultados computacionais obtidos é possível verificar que o ILS apresenta melhores soluções em relação aos demais métodos em 61,20% dos problemas, sendo que em 9,77% existe empate entre o

ILS e o AG-BL. Para os problemas com até 30 tarefas e 12 estágios, o AG-BL apresenta-se bastante competitivo em relação ao ILS. Mas nos problemas com 30 tarefas e 15 estágios, em diante, ILS mostrou-se superior.

Finalmente, pode-se concluir que o ILS utilizando a heurística construtiva NEH-MST, implementado nesta dissertação, apresenta bom desempenho em termos de qualidade de soluções obtidas.

O problema de programação *flowshop* flexível não se limita aos métodos que foram apresentados neste trabalho. Sugere-se a implementação do *path relinking*, integrado aos métodos ILS e GRASP. Outros estudos devem ser realizados, com objetivo de inserção de tempo ocioso ao problema.

REFERÊNCIAS BIBLIOGRÁFICAS

AARTS, E.; LENSTRA, J.K. **Local search in combinatorial optimization.** Eindhoven y Atlanta: Princeton University Press, 2003, p. 01-17.

ALLAHVERDI, A.; GUPTA, J.N.D.; ALDOWAISAN, T. **A review of scheduling research involving setup considerations.** OMEGA, The international Journal of Management Science, 27(2):219–239, 1999.

ALLAHVERDI, A.; NG, C.T.; CHENG, T.C.E.; KOVALYOV, M.Y. **A survey of scheduling problems with setup time or cost.** European Journal of Operational Research, 187, 2008. p. 985–1032.

ARENALES, M; ARMENTANO, V.; MORABITO, R.; YANASSE, H. **Pesquisa Operacional.** Rio de Janeiro: Elsevier, 2007. p. 15–43.

ARROYO, J.E.C. **Heurísticas e Metaheurísticas para otimização combinatória multiobjetivo.** Campinas: UNICAMP, 2002. 231 p. Tese (Doutorado) – Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, Campinas, 2002.

CHASE, R.B.; JACOBS, F.R.; AQUILANO, N.J. **Administração da Produção para a vantagem competitiva.** 10.ed, Porto Alegre: Bookman, 2006. p. 20–35.

COLIN, E. C. ***Beam search*** **e inserção de ociosidade no problema de programação de um máquina em ambiente do tipo Jit.** São Paulo: USP, 1997. 106 p. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo, São Paulo, 1997.

COLLETTE, Y.; SIARRY, P. **Multiobjective Optimization: Principles and Case Studies.** 1º ed. Paris: Springer, 2003. p. 4-7.

DAVOUDPOUR, H; ASHRAFI, M. **Solving multi-objective SDST flexible flow shop using GRASP algorithm.** The International Journal of Advanced Manufacturing Technol, 2009, p. 737 - 747.

DONG, X., HUANG, H. e CHEN, P. **An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion.** Computers & Operations Research , Elsevier, 36, 2009, p. 1664 – 1669.

DORIGO, M.; STUTZELE, T.. **Ant Colony Optimization.** Massachussetts Institute of Technology, EUA, 2004, p. 25-63.

FAKHRZAD, M.B.; HEYDAR, M. **A heuristic algorithm for hybrid flow-shop production scheduling to minimize the sum of the earliness andf tardiness cost.** Journal of the Chinese Institute of Industrial Engineers. Vol. 25, N° 2, 2008, p. 105-115.

GAREY, M.; JOHNSON,D. **Computers and intractability: a guide to the theory of NP completeness.** W.H. Freeman and Company, New York, 1999, p. 338.

GIGANTE, R.L. **Heurística construtiva para a programação de operações *flowshop* permutacional.** São Carlos: USP, 2010. 87 p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, São Carlos, 2010.

HASIJA, S., RAJENDRAN, C., 2004. **Scheduling in flowshops to minimize total tardiness of jobs**. International Journal of Production Research 42 (11), p. 2289–2301.

HILLIER, F.S.; LIEBERMAN, G.J. **Introduction to Operations Research.** 8.ed. New York: McGraw-Hill Publishing Company, 2004, p. 01-05.

HUTCHINS, D. **Just in Time.** Gower, 1999, p. 3-15.

JUNGWATTANAKIT, J; REODECHA, M, CHAOVALITWONGSE, P.; WERNER, F. **Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria.** International Journal of Advanced Manufacturing Technology, Vol. 37, 2008, p. 354 – 370.

KOGAN, K; KHMELNITSKY, E. **Scheduling: Control-based theory and polynomial-time algorithms.** Kluwer Academic Publishers, 2000, p. 3-19.

KURZ, M. E.; ASKIN, R. G. **Scheduling flexible flow lines with sequence-dependent setup times**. European Journal of Operational Research, v. 159, n.1,2004. p. 66-82.

LOGENDRAN, R.; DeSZOEKE, P.; BARNARD, F.; **Sequence-dependent group scheduling problems in flexible flow shops.** International Journal of Production Economics, 102, 2006, p. 66-86.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. **Iterated local search.** Handbook of Metaheuristics. Kluwer Academic Publishers, Boston, 2003, p. 321-353.

LUCENA, A; PONTES, R. **Aviação comercial controlada por máquinas inteligentes.** E-papers, Rio de Janeiro: E-papers, 2007, p. 15-23.

MACCARTHY, B.L.; LIU, J. **Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling.** International Journal of Production Research, v. 31, n.1, London, 1993. p.59-79.

MAINIERI, G.B. **Heurística para a minimização do atraso total no ambiente *flowshop* com múltiplos processadores.** São Paulo: UPS, 2009. 65 p. Dissertação (Mestrado) Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção, São Paulo, 2009.

MAINIERI, G.B.; RONCONI, D.P. **Uma Heurística para a Minimização do Atraso Total no Ambiente de *Flowshop* Flexível**. In: XL Simpósio Brasileirode Pesquisa Operacional, 2008, João Pessoa. Anais do XL SBPO. João Pessoa: SBPO, 2008. p. 1924 -1934.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**, 3ed. Springer-Verlag, Berlin, 1996. p. 13 – 22.

MONTGOMERY, D. **SDesign and Analysis of Experiments**. John Wiley & Sons, New York, 2007.

NADERI, B.; ZANDIEH, M.; FATEMI GHOMI, S.M.T. **Scheduling sequence-dependent setup time job shops with preventive maintenance.** International Journal of Advanced Manufacturing Technology 43, 2009, p. 170-181.

NAWAZ, M.; ENSCORE, E.; HAM, I. (1983). **A heuristic algorithm for the m-machine, n-job flowshop sequencing problem**. OMEGA, 11, 1983. p. 91 – 95.

OSMAN, I.H.; KELLY, J.P. **Meta-heuristics: theory & applications.** 2.ed, Boston: Kluwer Academic Publishers, 1996. p. 01 - 21.

PAPADIMITRIOU, C.H.; STEIGLITZ, K. **Combinatorial Optimization: algorithms and complexity.** Dover Publications, Mineola, New York, 1998.

PEINADO, J; GRAEML, A.R. **Administração da produção: operações industriais e de serviços**. Curitiba: UntcentP, 2007.

PINEDO, M. L. **Scheduling – theory, algorithms, and systems.** 3º ed. Springer, 2008.

RUIZ, R.; MAROTO, C. **A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility.** European Journal of Operational Research, 169, 2006. p. 781 – 800.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. **Two new robust genetic algorithms for the flowshop scheduling problem**. OMEGA, 34, 2006a. p. 461 – 476.

RUIZ, R.; SIVRIKAYA, F.; URLINGS, T. **Modeling realistic hybrid flexible flowshop scheduling problem.** Computers & Operational Research, 169, 2006b. p. 1151 – 1175.

RUIZ, R.; STÜTZLE, T. **An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives.** European Journal of Operational Research, 187, 2008. p. 1143–1159.

SOUZA, E. C. **Programação de tarefas em um flowshop.** São Paulo: USP, 2009. 124 p. Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Naval e Oceânica II, São Paulo, 2009.

VALLADA, E.; RUIZ, R. **Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times**. DEIOAC, 2009.

XHAFA, F.; ABRAHAM, A. **Metaheuristics for Scheduling in Industrial and Manufacturing Applications.** Springer, 2008. p. 11-24.