UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

ADRIANO SEIKO KOMESU

Heurística Evolutiva para a minimização do atraso total em ambiente de produção *flow shop* com
buffer zero

São Carlos

2015

ADRIANO SEIKO KOMESU

**HEURÍSTICA EVOLUTIVA PARA A MINIMIZAÇÃO DO ATRASO TOTAL EM
AMBIENTE DE PRODUÇÃO *FLOW SHOP* COM *BUFFER ZERO***

Dissertação apresentada à Escola de
Engenharia de São Carlos, da
Universidade de São Paulo para a obtenção
do título de Mestre em Engenharia de
Produção.

Área de Concentração: Processos e Gestão
de Operações.

Orientador: Prof. Dr. Marcelo Seido
Nagano

São Carlos
2015

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

KOMES
U, A.
S.h

Komesu, Adriano Seiko
Heurística Evolutiva para a minimização do atraso
total em ambiente de produção flow shop com buffer zero
/ Adriano Seiko Komesu; orientador Marcelo Seido
Nagano. São Carlos, .

Dissertação (Mestrado) - Programa de Pós-Graduação
em Engenharia de Produção e Área de Concentração em
Processos e Gestão de Operações -- Escola de Engenharia
de São Carlos da Universidade de São Paulo, .

1. Programação da Produção. 2. Buffer Zero. 3. Flow
Shop. 4. Métodos Heurísticos Evolutivos. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **ADRIANO SEIKO KOMESU**.

Título da dissertação: "Heurística evolutiva para a minimização do atraso total em ambiente de produção *flow shop* com *buffer zero*".

Data da defesa: 10/04/2015

Comissão Julgadora:

Prof. Dr. **Marcelo Seido Nagano (Orientador)**
(Escola de Engenharia de São Carlos/EESC)

Prof. Dr. **Walther Azzolini Júnior**
(Escola de Engenharia de São Carlos/EESC)

Profa. Dra. **Juliana Keiko Sagawa**
(Universidade Federal de São Carlos/UFSCar)

Resultado:

Aprovado

Aprovado

Aprovado

Coordenadora do Programa de Pós-Graduação em Engenharia de Produção:

Profa. Associada **Daisy Aparecida do Nascimento Rebelatto**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Paulo César Lima Segantine**

À minha família, com carinho e amor.

É justamente a possibilidade de realizar um sonho que torna a vida interessante.

(Paulo Coelho)

AGRADECIMENTOS

Ao Prof. Dr. Marcelo Seido Nagano pela orientação, dedicação, paciência e amizade.

Aos meus pais Seiko e Alice pelo apoio, carinho e dedicação.

Aos meus irmãos Andrea e Acácio pela amizade e apoio.

Aos meus amigos de longa data André, Péricles, Ana, Yudi e Pedro por sempre estarem presentes, mesmo longe.

Aos meus amigos de Faculdade, da Eletrônica 08, por todos os momentos vividos no período de faculdade que me motivaram a seguir com os estudos.

A Maria Fernanda, pela atenção, amor, carinho e dedicação.

A EESC-USP pela oportunidade de estudar em um lugar com excelência acadêmica.

A todos os meus professores que contribuíram para a minha formação educacional.

Resumo

KOMESU, A. S. **Heurística Evolutiva para a minimização do atraso total em ambiente de produção *Flow Shop* com *Buffer Zero***, 2015, 84p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015.

Este trabalho aborda o problema de programação de tarefas, a partir de um caso específico, conhecido como *Flow Shop* com *buffer zero*. O problema consiste em programar n tarefas em m máquinas no ambiente *Flow Shop* permutacional. Com o aumento do nível de exigência dos clientes, pesquisas que buscam o atendimento das datas de entrega têm se tornado de extrema importância em ambientes de manufatura. Este trabalho analisa o problema de minimização do atraso total no ambiente *flow shop* onde não existe a possibilidade de armazenagem das tarefas entre estágios de produção sucessivos (*buffer zero*), tendo como consequência o bloqueio de máquinas. A Heurística Evolutiva *Clustering Search* foi proposta e analisada para a obtenção de soluções de altíssima qualidade para o problema. Finalmente, uma extensa experimentação computacional foi realizada. Quando comparado com o melhor método reportado na literatura, o método proposto apresentou qualidade superior.

Palavras-chave: Programação da Produção, *Buffer Zero*, *Flow Shop*, Métodos Heurísticos Evolutivos.

Abstract

KOMESU, A. S. **Evolutionary Heuristic for total tardiness minimization in Flow Shop environment with no Buffer**, 2015, 84p. Thesis (MS) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015.

This work deals with the Flow Shop scheduling problem. The objective is scheduling n jobs on m machines in the Permutation Flow Shop environment. With the increasing customer demand level, researches that aims the attendance of due dates have become extremely important in manufacturing process. This work studies the total tardiness minimization problem in the flow shop environment where there is no buffer storage between machines, resulting in the machine block. The Heuristic Evolutionary Clustering Search was proposed and analyzed to obtain high quality solutions to the problem. Finally, an extensive computational experiment was performed. When compared to the best method reported in the literature, the proposed method showed high quality.

Keywords: Production Scheduling, Buffer Zero, Flow Shop, Evolutionary Heuristic Methods.

LISTA DE SIGLAS

n – Número total de tarefas
 m – Número total de máquinas
 $d_{i,m}$ – Tempo de partida (saída) da tarefa i na máquina m
 T – Tempo de Atraso
due date – Date de entrega da tarefa
 $p_{i,m}$ – Tempo de processamento da tarefa i na máquina m
 C – Centro do *Cluster*
 r – Raio do *Cluster*
 β – Tipo de Busca Local
ME – Metaheurística
AI – Agrupador Iterativo
AA – Analisador de Agrupamento
AO – Algoritmo de Otimização
 λ – Densidade do *Cluster*
 P – População
Ind – Indivíduo da População
 σ – sequência de tarefas
 σ_i – Tarefa da posição i da sequência σ
TT - Total Tardiness
 C_{max} – Número máximo de *Clusters*
 γ – Porcentagem de melhores indivíduos da População
 α – Probabilidade de ocorrer a Busca Local
ind - Indivíduo
LB – Lower Bound
 F – fator de atraso das tarefas
 R - faixa de dispersão

LISTA DE FIGURAS

Figura 1: Esquema que relaciona as diversas classes de problemas de programação de operações em máquinas (adaptado de MacCarthy e Liu, 1993).	25
Figura 2: Exemplo numérico para obtenção dos valores de partida de cada tarefa em cada máquina da Tabela 1 (Komesu, A. S. 2012).	38
Figura 3: Diagrama Conceitual do CS (Ribeiro Filho et al., 2007b).	40
Figura 4: Fluxograma Geral do Método ECS.	42
Figura 5 - Recombinação BOX (Adaptado de Ribeiro Filho et al., 2007b).	44
Figura 6: Gráfico das RDI das condições iniciais.	59
Figura 7: Gráfico dos valores de RDI por porte de problema do método ECS.	61
Figura 8: Gráfico da média dos valores de RDI de cada Cenário utilizando o método ECS.	61
Figura 9: Gráfico dos valores de RDI por porte de problema do método ILS.	63
Figura 10: Gráfico da média dos valores de RDI de cada Cenário do método ILS.	63
Figura 11: Gráfico comparativo das Médias de RDI por porte de problema dos métodos ECS e ILS.	64
Figura 12: Gráfico comparativo das Médias de RDI por cenário dos métodos ECS e ILS.	64
Figura 13: Gráfico da Porcentagem de Sucesso do melhor resultado do Método ECS referente a Tabela 8.	65
Figura 14: Gráfico da Porcentagem de Sucesso do melhor resultado do Método ILS referente a Tabela 9.	66
Figura 15: Gráfico do confronto entre as médias das porcentagens de sucesso entre os melhores resultados.	67
Figura 16: Gráfico da Porcentagem de Sucesso da média dos resultados do Método ECS referente a Tabela 10.	68
Figura 17: Gráfico da Porcentagem de Sucesso do Método ILS referente a Tabela 11.	69
Figura 18: Gráfico do confronto entre as médias das porcentagens de sucesso entre a média dos resultados.	70
Figura 19: Gráfico da média dos valores de RDI por porte de problema.	71

LISTA DE TABELAS

Tabela 1 - Exemplo numérico para obtenção dos valores de partida de cada tarefa em cada máquina (Komesu, A. S. 2012)	37
Tabela 2: Testes para Valores de C_{max}	56
Tabela 3: Tabela de variações dos Parâmetros r , α e γ	57
Tabela 4: Resultado da Tabela 3	58
Tabela 5: Melhores Parâmetros	59
Tabela 6: Valores de RDI do método ECS	60
Tabela 7: Valores de RDI do método ILS	62
Tabela 8: Porcentagem de Sucesso do método ECS para os melhores resultados	65
Tabela 9: Porcentagem de Sucesso do método ILS para os melhores resultados	66
Tabela 10: Porcentagem de Sucesso do método ECS para a média dos resultados	68
Tabela 11: Porcentagem de Sucesso do método ILS para a média dos resultados.....	69
Tabela 12: Valores da média de RDI dos métodos ECS e ILS por porte de problema	71
Tabela 13: Valores de <i>Best-Know</i> atualizada para o critério de <i>Total Tardiness</i>	72

SUMÁRIO

1. INTRODUÇÃO	23
1.1. Programação de operações em máquinas	24
1.2. Motivação e Justificativa para execução da pesquisa	25
1.3. Objetivos da pesquisa	25
2. PROBLEMA DE PROGRAMAÇÃO <i>FLOW SHOP</i> COM BLOQUEIO DE MÁQUINAS	27
3. HEURÍSTICA <i>CLUSTERING SEARCH</i>	39
3.1. Heurística Evolutiva <i>Clustering Search</i> para o Problema <i>Flow Shop</i> com <i>Buffer Zero</i>	41
3.1.1. Nawaz-Enscore-Ham (NEH)	46
3.1.2. Vizinhança de Inserção e Vizinhança de Permutação	47
3.1.2.1. Vizinhança de Permutação	47
3.1.2.2. Vizinhança de Inserção	47
3.1.3. Procedimento de Busca Local (VNS)	48
3.1.4. Procedimento LS1	49
3.1.5. Procedimento LS2	49
4. <i>ITERATED LOCAL SEARCH</i> COM <i>VARIABLE NEIGHBOURHOOD SEARCH</i>	51
4.1. Solução Inicial	51
4.2. Busca Local	52
4.3. <i>Acceptance Criterion</i>	52
4.4. Mecanismo de Perturbação	52
4.5. Ajuste dos parâmetros do algoritmo ILS	54

5.	EXPERIMENTAÇÃO COMPUTACIONAL	55
5.1.	Ajustes experimentais dos parâmetros do método ECS	56
5.2.	Análise dos Resultados	60
6.	CONSIDERAÇÕES FINAIS	75
6.1.	Sugestões para trabalhos futuros	76

1. Introdução

No atual cenário de produção e consumo globalizados, a disputa industrial torna-se cada vez mais acirrada, ocasionando uma busca intensa por maior eficiência dos métodos de produção e das técnicas de gerenciamento da atividade industrial. Objetiva-se diferenciar os produtos e serviços prestados de forma a torná-los mais atrativos ao público consumidor. Inseridos neste contexto, o estudo e aperfeiçoamento dos sistemas de produção passaram a ser fundamentais para as organizações se manterem atualizadas e competitivas.

Este trabalho tem como objetivo central o desenvolvimento de um método heurístico evolutivo e ferramentas computacionais para auxiliar o planejamento e programação de tarefas em sistemas de produção. Este tipo de pesquisa se insere na área de Planejamento e Controle da Produção (PCP) e trata dois de seus principais problemas:

- Planejamento da Produção;
- Programação da Produção.

Neste trabalho objetiva-se investigar o problema de Programação da Produção em ambiente *flow shop* de produção intermitente repetitiva, nos quais não existe a possibilidade de armazenagem das tarefas entre estágios de produção sucessivos (*buffer zero*). Tal condição tem como consequência o bloqueio de máquinas.

Tais ambientes de produção constituem segmentos relevantes como, por exemplo, as indústrias metalúrgicas, de produtos químicos, e farmacêuticos. Em determinadas situações, com o objetivo de aumentar a produtividade reduzindo custos, determinados tipos de sistemas produtivos podem ser organizados de forma a reduzir ou mesmo, se possível, eliminar espaços de armazenagem de tarefas entre estágios sucessivos de produção (*buffer zero*). Em outras situações, na presença de dispositivos automatizados de produção, uma tarefa deve permanecer na máquina atual até que a máquina da operação subsequente esteja liberada, causando o bloqueio da máquina atual.

1.1. Programação de operações em máquinas

Os Sistemas de Produção podem ser classificados de diversas formas. Johnson e Montgomery (1974) sugerem uma classificação em três categorias, segundo o processo de fluxo de materiais:

- a) Sistema Grande Projeto: tem como característica a produção de itens complexos e/ou de grande porte, na maioria dos casos com lotes unitários;
- b) Sistema Contínuo: tem como característica a produção em larga escala de produtos padronizados e com pouca diversificação;
- c) Sistema Intermitente: tem como característica a flexibilidade, ou seja, a capacidade de produzir um grande número de produtos. Este sistema é dividido em dois tipos:

- c.1) *Flow Shop*: os itens fabricados em uma linha ou célula de manufatura tem a mesma sequência de operações nas diversas máquinas;

- c.2) *Job Shop*: a sequência de operações se modifica de um produto para outro.

No contexto de um problema de programação da produção, as restrições tecnológicas e a medida de desempenho da programação devem ser especificadas. As restrições tecnológicas são determinadas principalmente pelo fluxo das operações das diversas tarefas nas máquinas. Neste aspecto, os problemas de programação de operações nas máquinas podem ser classificados da seguinte forma de acordo com MacCarthy & Liu (1993):

- a) *Job Shop*: cada tarefa tem sua própria sequência de processamento nas máquinas;
- b) *Flow Shop*: todas as tarefas têm a mesma sequência de processamento nas máquinas;
- c) *Open Shop*: não há uma sequência específica ou preestabelecida para as tarefas serem processadas nas máquinas;
- d) *Flow Shop Permutacional*: é um *Flow Shop* no qual em cada máquina a sequência das tarefas é a mesma;
- e) *Máquina Única*: existe uma única máquina disponível para o processamento das tarefas;
- f) *Máquinas Paralelas*: estão disponíveis diversas máquinas, geralmente idênticas, para as mesmas operações;
- g) *Job Shop com Máquinas Múltiplas*: é um *Job Shop* no qual existe um conjunto de máquinas paralelas em cada estágio de produção;

h) *Flow Shop com Múltiplas Máquinas*: é um *Flow Shop* no qual existe um conjunto de máquinas paralelas em cada estágio de produção.

A Figura 1 ilustra a relação entre as diversas classes dos problemas de programação de operações em máquinas.

Figura 1: Esquema que relaciona as diversas classes de problemas de programação de operações em máquinas (adaptado de MacCarthy e Liu, 1993).

1.2.Motivação e Justificativa para execução da pesquisa

A motivação para desenvolver a pesquisa proposta neste trabalho advém da tendência de investigação do problema proposto observada no exame da literatura para a proposição de métodos de soluções heurísticos e meta-heurísticos para a obtenção de soluções de altíssima qualidade para o problema.

1.3.Objetivos da pesquisa

Tendo por base os resultados obtidos em pesquisas anteriores e o exame da literatura sobre o problema objeto deste trabalho de pesquisa, os objetivos propostos são:

- Estudar o problema de Programação da Produção em ambiente *flow shop* de produção intermitente repetitiva, com máquinas (células / instalações / equipamentos) para sistemas produtivos nos quais não existe a possibilidade de

armazenagem das tarefas entre estágios de produção sucessivos (*buffer zero*), tendo como consequência o bloqueio de máquinas;

- Desenvolver método para solução do problema com boa qualidade de solução e eficiente computacionalmente aplicando a Heurística Evolutiva *Clustering Search*;
- Promover a divulgação dos resultados obtidos junto à comunidade científica nacional e internacional.

Tendo os objetivos propostos deste trabalho, o capítulo seguinte apresentará uma breve revisão da literatura do problema *flow shop* com bloqueio.

2. Problema de Programação *Flow Shop* com Bloqueio de Máquinas

No problema objeto desta pesquisa, o bloqueio de máquina deverá ocorrer sempre que a máquina subsequente não estiver disponível, ou seja, trata-se do caso conhecido como *buffer* zero. Esse tipo de sistema produtivo pode existir tanto em ambientes *Job Shops* quanto em *Flow Shops*.

Dentre os trabalhos mais importantes e pioneiros que foram reportados na literatura do sistema produtivo no ambiente *flow shop*, pode ser citado o artigo de Reddi e Ramamoorthy (1972), onde os autores mostram que o problema de programação *flow shop* tradicional com somente duas máquinas (dois estágios de produção) e a restrição de bloqueio de máquinas pode ser transformado em um caso especial do Problema do Caixeiro-Viajante (*TSP - Traveling Salesman Problems*).

Papadimitriou e Kanellakis (1980) provaram que o problema com capacidade de armazenagem de somente uma tarefa (*buffer* limitado) entre operações sucessivas é NP-difícil.

A intensificação de trabalhos sobre *flow shop* com bloqueio de máquinas ocorreu a partir do final dos anos 1980.

McCormick et al. (1989) apresentaram um método heurístico, denominado *Profile Fitting (PF)*, o qual procura sequenciar as tarefas de forma a minimizar a soma dos tempos de máquina parada, assim como dos tempos de bloqueio das máquinas.

Leisten (1990) apresentou dois métodos heurísticos para problemas *flow shop* permutacionais e não-permutacionais com *buffer* limitado. A medida de desempenho era maximizar a utilização dos *buffers* e minimizar o tempo de bloqueio das máquinas. Uma experimentação computacional mostrou que uma adaptação do tradicional método NEH (Nawaz et al., 1983), originalmente desenvolvido para o problema com *buffer* ilimitado, teve um desempenho melhor que os métodos propostos.

Hall e Sriskandarajah (1996), com base no resultado obtido por Papadimitriou e Kanellakis (1980), mostraram que o problema *flow shop* tradicional com três estágios de produção e restrição de bloqueio de máquinas é um problema fortemente NP-completo. Nesse mesmo artigo, os autores relatam os principais trabalhos já então reportados na literatura.

Nowicki (1999) desenvolveu uma meta-heurística de Busca Tabu que utiliza uma generalização da noção de bloco de tarefas que foi sugerida por Grabowski et al. (1983).

Abadi et al. (2000) propuseram um método heurístico para minimizar o tempo de ciclo em um *flow shop* com bloqueio de máquinas, utilizando a correlação entre este problema e o problema de programação *flow shop* com restrições *no-wait* na execução das tarefas.

Armentano e Ronconi (2000) apresentaram o problema *flow shop* com bloqueio com o critério de minimização do tempo total de atraso através da Busca Tabu, um dos primeiros trabalhos a apresentar solução para este tipo de problema.

Ronconi e Armentano (2001) apresentaram algoritmos *Branch-and-Bound* para a solução do problema. Foi proposto um novo limitante inferior (*lower bound*) para as datas de término das tarefas. A partir daí, foram obtidos limitantes inferiores para o atraso total e o *makespan* (duração total da programação).

Caraffa et al. (2001) desenvolveram um Algoritmo Genético para problemas *flow shop* de grande porte com restrições de fluxo (na literatura conhecidos como *Restricted Slowdown Flowshop Problems*), para os quais o problema com bloqueio de máquinas foi um caso especial.

Ronconi (2004) abordou o problema com capacidade zero de armazenagem de tarefas entre operações sucessivas (*buffer zero*), com o objetivo de minimizar o *makespan*. Um método heurístico construtivo que utiliza características específicas do problema foi apresentado. O novo método, combinado com outros existentes na literatura, foi comparado com o conhecido método heurístico NEH (Nawaz et al., 1983) originalmente desenvolvido para o problema de *flow shop* com *buffer* ilimitado, apresentando desempenho superior.

Ronconi (2005) apresentou um algoritmo *Branch-and-Bound*, no qual foram utilizados novos *lower bounds*, os quais utilizam de maneira vantajosa a natureza e estrutura do problema com bloqueio de máquinas. O desempenho do novo método foi melhor do que aqueles propostos em Ronconi e Armentano (2001).

Martinez et al. (2006) trataram do problema *flow shop* tradicional com *buffer zero* e com o objetivo de minimizar o *makespan*. Os autores apresentaram um novo tipo de restrição de bloqueio de máquinas (a máquina permanece bloqueada por um trabalho até que o trabalho na máquina seguinte seja terminado e ele saia da máquina seguinte), o qual utilizado com a restrição clássica (a máquina permanece bloqueada por um trabalho até que seja iniciado o processamento deste na próxima máquina) proporcionou a solução de problemas específicos de quatro estágios de produção em tempo polinomial.

Bagchi et al. (2006) realizaram uma revisão dos métodos baseados no *TSP* para solucionar problemas de *flow shop*. Os autores demonstram que diversos problemas de *flow*

shop, mesmo que bastante complexos, podem ser modelados como problemas de *TSP* e então solucionados por algoritmos disponíveis para eles.

Companys e Mateo (2007) estudaram dois problemas, o *flow shop* permutacional e o *flow shop* permutacional com bloqueio, ambos com o objetivo de minimizar o *makespan*. Inicialmente, os autores sugerem a heurística denominada NEH+, que é a aplicação de uma busca local conhecida por *NEDM-RCT* (*Non-Exhaustive Descent Method with Random Consideration of Ties*) juntamente com a heurística NEH (Nawaz *et al.*, 1983). Quando testado por meio de uma base de dados criada pelos autores, a aplicação da busca local melhorou significativamente os resultados tanto para problemas de *flow shop* permutacional como para problemas de *flow shop* permutacional com bloqueio. Em seguida foi proposto o algoritmo de duplo *branch-and-bound* *LOMPEN* (*Lomnicki Pendular algorithm*), proposto por Companys (1993), usando a heurística NEH+ para obter a solução inicial. Por fim, foi proposto o uso de 10 diferentes heurísticas para compor a solução inicial para o *LOMPEN*. Para os testes computacionais foram utilizados os problemas-teste de Taillard (1993) e outro conjunto de problemas-teste criada pelos autores.

Grabowski e Pempera (2007) desenvolveram meta-heurísticas de Busca Tabu (*TS – Tabu Search*) para a minimização do *makespan*. Algumas propriedades relacionadas ao bloqueio das máquinas foram apresentadas, discutidas e utilizadas nos métodos propostos. Experimentalmente, os autores mostraram que os métodos propostos tiveram um desempenho relativo superior na busca de melhores soluções, com pequeno esforço computacional, quando comparados com métodos já reportados na literatura.

Ronconi e Henriques (2009) trataram do problema de *flow shop* com bloqueio de máquinas (*buffer zero*), com o objetivo de minimizar o atraso total na execução das tarefas. Além de um método heurístico construtivo que utiliza características específicas do problema, foi apresentado um método *GRASP* (*Greedy Randomized Adaptive Search Procedure*). Foram efetuados testes computacionais envolvendo os métodos propostos, uma adaptação do clássico método NEH (Nawaz *et al.*, 1983) e um algoritmo *branch-and-bound*. Os resultados experimentais mostraram que os novos métodos são promissores.

Liu e Kozan (2009) trataram do problema de *flow shop* com bloqueio combinado, com objetivo de minimizar o *makespan*. No problema tratado, as condições de *buffer* entre cada uma das máquinas é diferente (*buffer limitado*, *buffer zero* e *buffer ilimitado*). Inicialmente foi proposta pelos autores uma heurística construtiva denominada Liu-Kozan para solucionar o problema. Posteriormente a heurística construtiva foi combinado ao método NEH (Nawaz *et al.*, 1983) para melhorar os resultados. Os métodos propostos foram capazes de gerar

resultados para o problema, mostrando-se bastante eficientes, por conseguir tratar de problemas com diferentes tipos de *buffer*.

Wang et al. (2010a) propuseram um algoritmo evolutivo híbrido *HDDE* (*Hybrid Discrete Differential Evolution*), com o objetivo de minimização do *makespan*. Por meio de uma experimentação computacional, utilizando os problemas-teste de Taillard (1993), os autores mostraram que o algoritmo proposto obtém melhores soluções do que as meta-heurísticas de Busca Tabu e Busca Tabu com multi-movimentos (*TS+M – Tabu Search with Multi-Moves*) apresentados por Grabowski e Pempera (2007), como também superam o algoritmo *HDE* (*Hybrid Differential Evolution*) desenvolvido por Qian et al. (2009) em termos de qualidade de solução, robustez e eficiência de busca. Além disso, 112 das 120 melhores soluções obtidas por Grabowski e Pempera (2007) e Ronconi (2005) são melhoradas pelo algoritmo híbrido *HDDE* proposto.

Wang et al. (2010b) apresentaram três algoritmos híbridos *HS* (*Harmony Search*) que foram chamados por *hHS*, *hgHS*, e *hmgHS*, para o problema *flow shop* com bloqueio de máquinas e minimização do tempo total de fluxo (ou, equivalentemente, o tempo médio de fluxo). Originalmente, o algoritmo *HS* foi proposto por Geem, Kim, e Loganathan (2001), sendo considerado como a mais recente extensão dos Algoritmos Genéticos, apresentando simplicidade matemática e evolução (convergência para as melhores soluções) mais rápida. Os métodos propostos foram avaliados e comparados por meio de uma extensa experimentação computacional, utilizando problemas-teste reportados na literatura. Os resultados experimentais mostraram que os métodos propostos apresentam boas soluções, especialmente o algoritmo *hmgHS*.

Gong et al. (2010) trataram do problema *flow shop* com duas máquinas, onde a primeira máquina produz por lotes e a segunda produz individualmente, com bloqueio e *setup* compartilhado e não antecipatório. A função objetivo do problema foi minimizar a soma ponderada do *makespan* e do tempo de bloqueio. Inicialmente os autores apresentam um modelo em programação quadrática inteira mista (*QMIP – Quadratic Mixed Integer Program*) para a solução do problema e comprovaram que o problema é fortemente NP-difícil. Em seguida são apresentadas pelos autores duas heurísticas baseadas em regras para solucionar duas variantes do problema principal (um caso onde não há bloqueio e outro onde a sequência é predeterminada). Para os testes foram gerados aleatoriamente 2640 problemas, alterando o número de tarefas, o tamanho do lote, o tempo de *setup* e o fator de peso. Os problemas de pequeno porte foram resolvidos com o *QMIP* e com as heurísticas, já os problemas maiores

foram solucionados apenas com as heurísticas. As heurísticas apresentadas foram capazes de gerar bons resultados, apresentando também eficiência computacional.

Em 2011, os mesmos autores de Wang et al. (2010b) adaptaram o algoritmo *hmgHS* (*hybrid modified globalbest Harmony Search*) mencionado no artigo anterior, agora para minimização do *makespan* em um *flow shop* com bloqueio de máquinas. Por meio de simulação computacional e comparação com outros métodos existentes, os autores mostraram a superioridade do método proposto em termos de qualidade da solução.

Ribas et al. (2011) apresentaram um algoritmo iterativo *greedy* (“guloso”) para minimização do *makespan* em um *flow shop* com bloqueio de máquinas. Além do algoritmo *IG* (*Iterated Greedy*), os autores apresentaram um método heurístico de melhoria baseado no método NEH (Nawaz et al., 1983), o qual foi utilizado como procedimento de solução inicial para o algoritmo *IG*. Os desempenhos dos métodos apresentados foram avaliados utilizando os problemas-teste de Taillard (1993). Os resultados experimentais mostraram a eficiência do método proposto quando comparado com os melhores métodos reportados na literatura. As soluções de alguns problemas-teste utilizados na experimentação computacional foram melhoradas.

Pan et al. (2011a) apresentaram um algoritmo híbrido *HDDE* (*Hybrid Discrete Differential Evolution*) associado às buscas locais de inserção e permutação para solucionar problemas de *flow shop* com *buffer* intermediário, com objetivo de minimizar o *makespan*. Utilizando os problemas-teste de Taillard (1993) o algoritmo foi comparado com os métodos *P_DDE* proposto por Pan et al. (2008), ao *W_DDE* apresentado por Wang et al. (2010), ao *HGA* (*Hybrid Genetic Algorithm*) proposto por Wang et al. (2006) e ao *HPSO* (*Hybrid Particle Swarm Optimization*) proposto por Liu et al. (2008). O algoritmo obteve ótimos resultados, principalmente para problemas maiores, e com tempos computacionais próximos aos demais métodos comparados.

Ainda em 2011 Pan et al. (2011b) foi proposto um híbrido *HS* (*Harmony Search*) denominado *CHS* (*Chaotic Harmony Search*) para solucionar problemas de *flow shop* permutacional com *buffer* limitado, com o objetivo de minimizar o *makespan*. O método proposto foi avaliado e comparado com os algoritmos *IHS* (*Improved Harmony Search*), *GHS* (*Global best Harmony Search*), *ACO* (*Ant Colony Optimization*), *HPSO* (*Hybrid Particle Swarm Optimization*) e o *HGA* (*Hybrid Genetic Algorithm*). Os resultados experimentais mostraram que o método proposto apresentou melhores soluções.

Pan e Wang (2012) trataram do problema tradicional de programação *flow shop* com bloqueio de máquinas, com o objetivo de minimizar o *makespan*. Inicialmente, os autores

apresentam dois métodos heurísticos construtivos simples, denominados *wPF* e *PW*, ambos baseados no procedimento *PF* (*Profile Fitting*) introduzido por McCormick et al. (1989), e explorando as características do problema de bloqueio. Em seguida foram propostos três métodos heurísticos construtivos de “melhoria”, denominados *PF-NEH*, *wPF-NEH*, e *PW-NEH*, combinando os métodos inicialmente mencionados com o procedimento de enumeração do clássico método *NEH* (Nawaz et al., 1983). Finalmente, utilizando o método de busca local baseado em inserção de tarefas (*LS – Local Search*) são desenvolvidos três métodos heurísticos “compostos” que foram denotados por *PF-NEH/LS*, *wPF-NEH/LS*, e *PW-NEH/LS*. Os novos métodos propostos foram avaliados e comparados com os métodos existentes por meio de uma simulação computacional utilizando os problemas-teste de Taillard (1993). Os resultados experimentais mostraram que os métodos construtivos apresentados tiveram um desempenho bem melhor do que os métodos já reportados na literatura. Além disso, os métodos compostos conseguiram melhorar de maneira significativa a qualidade das soluções obtidas pelos construtivos. A experimentação computacional forneceu melhores soluções para 17 problemas-teste de grande porte, do banco de dados utilizado.

Trabelsi et al (2012) apresentaram uma heurística *TSS* (*Trabelsi, Sauvey e Sauer*) para solucionar problemas de *flow shop* com bloqueio misto, com o objetivo de minimizar o *makespan*. Os autores apresentaram também uma adaptação da heurística *NEH* para o problema. Ambos os métodos foram combinados com a meta-heurística *GA* (*Genetic Algorithm*) para solucionar os 100 problemas gerados pelos autores. Os resultados obtidos com as heurísticas em conjunto com o *GA* foram comparados com as já conhecidas soluções ótimas dos problemas e obtiveram bons resultados com bons tempos computacionais.

Han et al. (2012) propuseram um algoritmo *DABC* (*Discrete Artificial Bee Colony*) para minimizar o tempo total de fluxo em problema *flow shop* com bloqueio. Para a construção da solução inicial os autores utilizaram o *MME* (Ronconi, 2004). Ao *DABC* foi aplicada na etapa das abelhas oportunistas uma estratégia de busca local de inserção e na etapa das abelhas exploradoras uma estratégia de destruição e reconstrução. O algoritmo foi testado e comparado com três algoritmos propostos por Wang et al. (2010): *hHS* (*hybrid Harmony Search*); *hgHS* (*hybrid global-best Harmony Search*); e *hmgHS* (*hybrid modified global-best Harmony Search*). Para a avaliação dos métodos foram utilizados os problemas-teste de Taillard (1993). Os algoritmos obtiveram resultados melhores que os demais métodos.

Guanlong et al. (2012) trataram do problema de *flow shop* com bloqueio com o objetivo de minimizar o tempo total de fluxo. O algoritmo apresentado pelos autores foi uma

modificação do *DABC* (*Discrete Artificial Bee Colony*). O algoritmo foi testado e comparado com outros três algoritmos: *DABC* (*Discrete Artificial Bee Colony*) proposto por Tasgetiren et al. (2011), *HDDE* (*Hybrid Discrete Differential Evolution*) e *IG* (*Iterated Greedy*), utilizando os problemas-teste de Taillard (1993). Os resultados mostraram que o *DABC* proposto por Guanlong et al. (2012) foi significativamente melhor que os demais métodos com os quais ele foi comparado.

Wang e Tang (2012) adotaram um controle de diversidade auto-adaptativo ao *DPSO* (*Discrete Particle Swarm Optimization*). Além disso, um método de busca local conhecido como *stochastic variable neighborhood search* foi utilizado para intensificar a busca. O algoritmo foi então testado e comparado com o algoritmo busca tabu proposto por Grabowski e Pempera (2007), o algoritmo genético de Caraffa et al. (2001) e o *branch-and-bound* de Ronconi (2005) utilizando os problemas-teste de Taillard (1993). Os resultados mostraram que o método proposto foi superior aos métodos comparados, obtendo 111 melhores limitantes superiores (*upper bounds*) dos 120 problemas avaliados.

Han et al. (2012a) desenvolveram um algoritmo *IABC* (*Improved Artificial Bee Colony*) para problemas de *flow shop* com bloqueio e com o objetivo de minimizar o *makespan*. O algoritmo foi comparado com o *DABC* (*Discrete Artificial Bee Colony*) de Guanlong et al. (2012) e com o *HDDE* (*Hybrid Discrete Differential Evolution*) de Pan et al. (2012a) utilizando para avaliação os problemas-teste de Taillard (1993). A experimentação computacional apresentou a superioridade do método proposto na qualidade das soluções obtidas e no seu processamento computacional.

Bautista et al. (2012) adaptaram o uso do *BDP* (*Bounded Dynamic Programming*) para minimizar o *makespan* em problemas de *flow shop* permutacional com bloqueio. O método proposto foi testado utilizando os problemas-teste de Taillard (1993) e foi capaz de melhorar 17 das melhores soluções conhecidas dentre os 120 problemas da base de dados. Dentre os problemas que tiveram os resultados melhorados estão os 10 problemas de 500 tarefas e 20 máquinas (os maiores da base de dados).

Wang et al. (2012) apresentaram um *TPA* (*Three-Phase Algorithm*) para o problema de *flow shop* com bloqueio, com o objetivo de minimizar o *makespan*. Na primeira fase do algoritmo é aplicada uma regra de prioridade para organizar a solução inicial, em uma segunda fase o resultado é melhorado com a aplicação do NEH modificado, por fim é aplicado o algoritmo *SA* (*Simulated Annealing*) modificado. Os resultados obtidos com o método foram comparados com os obtidos pelo *HDDE* (*Hybrid Discrete Differential Evolution*) e pelos métodos apresentados por Ribas et al. (2011). O *TPA* superou significativamente os demais

métodos e ainda foi capaz de definir novos e melhores *upper bounds* para 53 dos 60 problemas de grande porte dos problemas-teste de Taillard (1993).

Davendra e Bialic-Davendra (2013) apresentaram uma metaheurística *DSOMA* (*Discrete Self-Organising Migrating Algorithm*) para minimizar o *makespan* em problemas de *flow shop* permutacional com bloqueio. O algoritmo foi comparado com os até então três melhores métodos para o problema. Os testes foram realizados utilizando duas bases de dados diferentes, comprovando o melhor desempenho do método proposto. O método foi capaz ainda de identificar novos *upper bounds* para diversos problemas-teste de Taillard (1993).

Recentemente, Ribas et al. (2013) propuseram uma busca local *ILS* (*Iterated Local Search*) combinada a um mecanismo de perturbação *VNS* (*Variable Neighbourhood Search*) para solucionar problemas *flow shop* permutacionais com bloqueio e com o objetivo de minimizar o tempo total de atraso. Inicialmente os autores trabalharam a solução inicial, propondo dois híbridos do NEH: um deles utilizando a regra *EDD* (*Earliest Due Date*) e o outro utilizando a regra *FPD* (*Fitting Processing times and Due dates*). Ambos resultaram em resultados similares. Em seguida foram testadas as buscas locais considerando apenas uma vizinhança (inserção ou permutação) e a busca local considerando as duas vizinhanças. A busca local considerando apenas a inserção obteve melhores resultados que a busca local considerando apenas a permutação, porém a busca local que utiliza as duas vizinhanças obteve resultados superiores aos dois. Finalmente o *ILS* foi comparado ao *GRASP* (*Greedy Randomized Adaptive Search Procedure*) de Ronconi e Henriques (2009) e o *IG* (*Iterated Greedy*) de Ribas et al. (2011). Os resultados comprovaram que o algoritmo apresentado obtém os melhores resultados, e, ainda, quanto maior e mais complexo é o problema, maior é a diferença entre os resultados obtidos pelo *ILS* combinado com o *VNS* em comparação com os demais métodos avaliados.

Lin e Ying (2013) propuseram um algoritmo *RAIS* (*Revised Artificial Immune System*) para problemas de *flow shop* permutacional com bloqueio com objetivo de minimizar o *makespan*. O algoritmo foi comparado a 6 outras metaheurísticas, obtendo melhores resultados, identificando ainda novos melhores resultados para os problemas-teste de Taillard (1993).

Moslehi e Khorasanian (2013) apresentaram dois modelos de programação inteira mista binária (*MIBP - Mixed Binary Integer Program*) para solucionar problemas de *flow shop* permutacional com bloqueio, com o objetivo de minimizar o tempo total de fluxo. Um dos modelos baseia-se no tempo de término de cada tarefa em cada máquina e o outro se baseia no tempo de ociosidade e de bloqueio das tarefas. Foram propostos três *lower bounds*

para o problema, os quais obtiveram resultados melhores que o *lower bound* proposto em estudos anteriores. O modelo apresentado foi capaz de encontrar a solução ótima de 30 problemas de Taillard (1993) com baixo tempo computacional.

Pan et al. (2013) apresentaram um algoritmo *MA* (*Memetic Algorithm*) de alto desempenho para o problema de *flow shop* com bloqueio para a otimização do valor de *makespan*. Para solução inicial foi utilizado o *PF* (*Profile Fitting*) juntamente com o NEH. O algoritmo foi comparado com as cinco melhores metaheurísticas conhecidas para o problema de *flow shop* com bloqueio: *HDDE* (*Hybrid Discrete Differential Evolution*); *IG* (*Iterated Greedy*); *HGA* (*Hybrid Genetic Algorithm*); *HPSO* (*Hybrid Particle Swarm Optimization*); *DPSO* (*Discrete Particle Swarm Optimization*). Foram ainda adaptados mais três outros algoritmos genéticos híbridos que eram utilizados para o problema clássico de *flow shop*. O algoritmo apresentado superou os demais e ainda melhorou, em 75 dos 120 problemas-teste de Taillard (1993) para os *upper bounds* fornecidos por Ribas et al. (2011).

Em um estudo mais recente, Han et al. (2013) investigaram o problema de *flow shop* com bloqueio e com o objetivo de minimizar o tempo total de fluxo. Inicialmente foram propostas duas variações da heurística *MME* (Ronconi, 2004), chamadas de *MME-A* e *MME-B*, para compor o quadro de soluções iniciais. Em seguida é apresentado um algoritmo híbrido *hDABC* (*hybrid Discrete Artificial Bee Colony*) para solucionar o problema. Por fim uma busca local de inserção é aplicada ao *hDABC* de três maneiras distintas: na etapa das abelhas trabalhadoras (*hDABC1*); na etapa das abelhas oportunistas (*hDABC2*) e na fase das abelhas exploradoras (*hDABC3*). Os três algoritmos foram então comparados a três outros híbridos propostos por Wang et al. (2010): *hHS* (*hybrid Harmony Search*); *hgHS* (*hybrid global-best Harmony Search*); *hmgHS* (*hybrid modified global-best Harmony Search*). Os testes foram feitos utilizando os problemas-teste Taillard (1993) e apresentaram a superioridade dos algoritmos propostos em relação aos demais, tanto em qualidade de solução como em eficiência e robustez.

Companys e Ribas (2014) apresentaram uma nova heurística construtiva para o problema de *flow shop* com bloqueio com o critério de avaliação a minimização do valor total do atraso. A heurística apresentada baseia-se em um algoritmo de três fases: a primeira fase seleciona a primeira tarefa, a segunda fase organiza as tarefas restantes e a terceira fase utiliza o procedimento de inserção NEH para melhorar a sequência. Neste trabalho foram implementadas doze diferentes heurísticas (combinações baseadas na alteração das fases um e dois). A melhor heurística apresentada foi a que focou nos valores de espera e tempo de folga da tarefa.

Han, Yu-Yan et al. (2014) apresentou o método DE-ABC (*differential evolution artificial bee colony*) para o problema *flow shop* com bloqueio com o critério de avaliação de minimização do *makespan*. O método proposto apresenta três operadores chaves: um é o *employed bee operator* que seria uma adoção de mutação e *crossover*, o segundo seria relacionada a um espectador que realiza modifica as soluções usando operadores de inserção ou de *swap* e o último um procedimento de busca local que seria uma busca local baseada em vizinhança de inserção. O algoritmo DE-ABC mostrou superioridade em relação ao TS+M, $DPSO_{svns}$, DABC, HDDE, EDEc e IABC.

Xie, Zhanpeng et al. (2014) apresentaram o algoritmo híbrido (VNSSA) que incorpora o *simulated annealing* (SA) e o método *variable neighborhood search* (VNS) para a solução do problema *flow shop* com bloqueio com o critério de minimização do *flow time*. O algoritmo foi comparado com os métodos hHS, hgHS, hmgHS, DABC e apresentou superioridade em relação a eles.

Conforme a revisão bibliográfica apresentada, a literatura que trata do ambiente *flow shop* tradicional com bloqueio de máquinas é relativamente extensa. Entretanto, a partir da revisão da literatura para o mesmo problema com o objetivo de minimização do tempo total de atraso foram encontrados somente três trabalhos que foram publicados por Armentano e Ronconi (2000), Ronconi e Henriques (2009), Ribas et al. (2013) e Companys e Ribas (2014).

O problema *flow shop* com bloqueio utilizando o critério do tempo total de atraso é conhecido como: $Fm | block | \sum T$, conforme a notação proposta por Graham et al. (1979), e pode ser modelada utilizando as expressões a seguir:

$$d_{1,0} = 0 \quad (1)$$

$$d_{1,k} = \sum_{q=1}^k p_{1,q}, \quad k = 1, 2, \dots, m-1, \quad (2)$$

$$d_{i,0} = d_{i-1,1}, \quad i = 2, \dots, n, \quad (3)$$

$$d_{i,k} = \max(d_{i,k-1} + p_{i,k}, d_{i-1,k+1}), \quad i = 2, \dots, n, k = 1, \dots, m-1 \quad (4)$$

$$d_{i,m} = d_{i,m-1} + p_{i,m}, \quad i = 1, \dots, n, \quad (5)$$

A equação (1) define da data de início do processo, ou seja, a data de início da primeira tarefa na primeira máquina. A equação (2) calcula a data de partida (tempo em que a tarefa é

liberada da máquina) da primeira tarefa em todas as máquinas. A partir da equação (3), o equacionamento do problema torna-se um processo iterativo no qual o tempo de partida $d_{i,k}$ da equação (4) é o maior valor entre a soma do tempo de partida da tarefa anterior na máquina anterior com o tempo de processamento $p_{i,k}$ e o valor do tempo de partida da tarefa anterior na máquina posterior, ou seja, recursivamente, os tempos de partida da primeira tarefa são calculados em todas as máquinas primeiramente (equações 1 e 2), posteriormente os da segunda tarefa e assim sucessivamente até a última tarefa. Caso $d_{i-1,k+1}$ seja maior que $d_{i,k-1} + p_{i,k}$, a diferença entre estes valores, obtemos o valor de *blocking* (tempo de bloqueio) e caso o contrário, a diferença seria o valor de *idle* (tempo de espera).

A Tabela 1 apresentam valores de processamento de cinco tarefas em cinco máquinas. A Figura 2 ilustra como são alocadas as tarefas da Tabela 1 no processo de *flow shop* com *blocking* e *buffer* zero.

Tabela 1 - Exemplo numérico para obtenção dos valores de partida de cada tarefa em cada máquina (Komesu, A. S. 2012)

	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	J Tarefa 5
Máquina 1	2	5	6	1	9
Máquina 2	3	8	7	10	4
Máquina 3	4	9	3	12	6
Máquina 4	7	1	5	9	1
Máquina 5	2	2	6	8	4

Figura 2: Exemplo numérico para obtenção dos valores de partida de cada tarefa em cada máquina da Tabela 1 (Komesu, A. S. 2012)

Com os tempos totais de cada tarefa, podemos calcular o tempo total de atraso de acordo com a equação (6).

$$T = \sum_{i=1}^n \max(d_{i,m} - due\ date_i, 0) \quad (6)$$

Onde $due\ date_i$ é a data de entrega .

Finalizando a seção referente à revisão bibliográfica do problema de programação *flow shop* com bloqueio de máquinas, é apropriado notar que o problema que está sendo proposto para ser investigado nesta pesquisa ainda não foi intensivamente pesquisado e reportado na literatura.

Nos capítulos seguintes, serão apresentados: o método proposto para a solução do problema deste trabalho e o melhor método reportado na literatura atual. Após, serão apresentados os resultados da extensa experimentação computacional realizada para fins de comparação.

3. Heurística *Clustering Search*

Conforme apresentada inicialmente, a proposta deste trabalho é a aplicação da Heurística Evolutiva *Clustering Search* (CS) para a solução do problema de programação de operações *flow shop* com *buffer* zero.

As primeiras aplicações da Heurística Evolutiva *Clustering Search* (CS) foram propostas por Oliveira e Lorena (2004, 2007), e os primeiros trabalhos para o problema de programação *flow shop* foram desenvolvidos por Ribeiro Filho, Nagano e Lorena (2007a, 2007b) e Nagano, da Silva e Lorena (2012).

O CS consiste de um processo de agrupamentos de soluções para detectar regiões supostamente promissoras no espaço de busca. O objetivo de encontrar tais regiões assim que possível é adaptar a estratégia de busca sobre elas. Uma região pode ser vista como um subespaço de busca definido por uma relação de vizinhança.

No CS um processo de agrupamento iterativo é executado simultaneamente com uma metaheurística, identificando grupos de soluções que merecem especial interesse. As regiões destes grupos de soluções devem ser exploradas tão logo sejam detectadas, através de heurísticas específicas de busca local. Espera-se uma melhoria no processo de convergência associado a uma diminuição no esforço computacional em virtude do emprego mais racional dos métodos de busca local.

O CS procura localizar regiões promissoras através da representação destas regiões por *clusters*. Um *cluster* é definido pela tripla $G = \{C, r, \beta\}$ onde C , r e β são, respectivamente, o centro, raio de uma região de busca, e uma estratégia de busca associada ao *cluster*.

O centro C é uma solução que representa o *cluster*, identificando a sua localização dentro do espaço de busca, e pode ser alterado durante o processo. Inicialmente, os centros são obtidos aleatoriamente e progressivamente tendem a deslocar-se para pontos realmente promissores no espaço de busca. O raio r estabelece a distância máxima, a partir do centro, até a qual uma solução pode ser associada ao *cluster*. Por exemplo, r pode ser definido como o número de movimentos necessários para transformar uma solução em outra. A estratégia de busca β é uma sistemática de intensificação de busca, na qual soluções de um *cluster* interagem entre si, ao longo do processo de agrupamento, gerando novas soluções.

O CS consiste de quatro componentes conceitualmente independentes com diferentes atribuições: uma metaheurística (ME), um agrupador iterativo (AI), um analisador de agrupamentos (AA) e um algoritmo de otimização local (AO). A Figura 3 ilustra os quatro componentes, o conjunto de soluções e os centros dos *clusters*, que interagem entre si.

Figura 3: Diagrama Conceitual do CS (Ribeiro Filho et al., 2007b)

O componente ME trabalha como um gerador iterativo de soluções. O algoritmo é executado independentemente dos componentes restantes e precisa ser capaz de gerar soluções de forma contínua para o processo de agrupamento. Este processo trabalha como um laço finito (limitado por tempo ou número de iterações), no qual, soluções são geradas ao longo das iterações.

O componente AI objetiva reunir soluções similares dentro de *clusters*, mantendo uma solução no centro do *cluster* que seja representativa para as demais soluções. Para evitar um esforço computacional extra, AI é desenvolvido como um processo *online*, no qual os agrupamentos são progressivamente alimentados por soluções geradas em cada iteração de ME. Um número máximo de *clusters* MC é definido a priori, evitando a criação ilimitada de *clusters*. Uma métrica de distância também precisa ser definida inicialmente, permitindo uma medida de similaridade para o processo de agrupamento. Cada solução recebida por AI é agrupada no *cluster* que for mais similar a ela, causando uma perturbação em seu centro. Tal perturbação é chamada de assimilação e consiste basicamente em atualizar o centro com a nova solução recebida.

O componente AA provê uma análise de cada *cluster*, em intervalos regulares, indicando um provável *cluster* promissor. A densidade do *cluster*, λ_i , é a medida que indica o nível de atividade dentro do *cluster*. Por simplicidade, λ_i pode ser o número de soluções

geradas por ME e alocadas para o *cluster* i . Sempre que λ_i atinja certo limitante, significando que algum modelo de informação foi predominantemente gerado por ME, tal *cluster* precisa ser mais bem investigado para acelerar o processo de convergência deste. O AA também é responsável por eliminar os clusters com baixa densidade, permitindo criar novos *clusters* e preservando os *clusters* mais ativos. A eliminação do *cluster* não afeta o conjunto de soluções de ME, somente o centro do *cluster* é considerado relevante para o processo.

Por fim, o componente AO é um módulo de busca local que provê a exploração de uma suposta região promissora, delimitada pelo *cluster*. Este processo acontece após AA ter descoberto um *cluster* promissor e uma busca local é aplicada no centro do *cluster*. AO pode ser considerado como a estratégia de busca β associada com o *cluster*, ou seja, uma heurística específica para o problema, que será empregada dentro do *cluster*.

3.1. Heurística Evolutiva *Clustering Search* para o Problema *Flow Shop* com *Buffer Zero*

Neste trabalho foram utilizadas heurísticas e metaheurísticas em conjunto para a solução do problema *flow shop* com *buffer zero*. Esta junção envolve métodos como NEH, *Variable Neighborhood Search* (VNS) e *Evolutionary Clustering Search* (ECS) que é a combinação do Algoritmo Genéticos (AG) e *Clustering Search* (CS). Para algoritmo ECS foram realizadas diversas modificações na concepção geral do método CS apresentado, abaixo, segue a representação geral do método.

Passo 1: Inicialização da População P

Passo 2: Inicialização do conjunto de *Clusters*.

Passo 3: Se *critério_de_parada* for falso, faça Passo 4, caso contrário, faça Passo 12.

Passo 4: Se $i < \text{novos_individuos}$, faça Passo 5, caso contrário, pule para o Passo 11.

Passo 5: pai1 = Selecione entre $\gamma\%$,melhores da População P

Passo 6: pai2 = Selecione um indivíduo qualquer em P.

Passo 7: Gere um *Filho* com a Recombinação(pai1 , pai2).

Passo 8: Execute a LS1 na sequência *Filho* com $\alpha\%$ de probabilidade.

Passo 9: Caso a inserção do *Filho* na população for verdadeira, realize o componente AI na sequência gerada *Filho* no conjunto de *Clusters* C.

Passo 10: Faça $i = i + 1$ e faça Passo 4.

Passo 11: Para cada *cluster* $c \in C$, caso seja verdadeira a execução do componente AA em c , execute o componente AO em c . Faça Passo 3.

Passo 12: O melhor indivíduo do processo é a solução do problema.

A Figura 4 apresenta um fluxograma macro da sequência do método ECS. Cada etapa do fluxograma abaixo será explorado no decorrer do texto.

Figura 4: Fluxograma Geral do Método ECS

Inicialmente, as tarefas foram ordenadas de forma não-decrescente de acordo com o tempo de *due date* e a partir desta ordenação, a sequência é construída utilizando a segunda etapa do método NEH. Em seguida, são criados novos indivíduos até que a população se totalize em 20 indivíduos e cada novo indivíduo é uma sequência que sofreu uma perturbação a partir da sequência construída pelo método NEH. Como a qualidade dos indivíduos da população inicial é de grande importância no resultado do componente AG e, para garantir esta qualidade, no processo de inicialização, os indivíduos são passados por um processo de busca local (VNS). Segue abaixo a perturbação realizada na etapa inicial do algoritmo.

Passo 1: Faça $length = \max(n/5, 30)$.

Passo 2: $j = 1$

Passo 3: Seja i um número aleatório entre 1 e $n-1$

Passo 4: Troque as tarefas de posição i e $i+1$ entre si.

Passo 5: Se $j = 2$, faça Passo 6, caso contrário, faça $j = j + 1$ e Passo 3.

Passo 6: Faça $l = 0$ e $w = length + 1$.

Passo 7: Se o absoluto($l - w$) $> length$ faça Passo 8, caso contrário faça Passo 9.

Passo 8: Faça $l =$ número aleatório entre 1 e n e $w =$ número aleatório entre 1 e n .e
faça Passo 7.

Passo 9: Troque as tarefas de posição l e w entre si.

A rotina de inicialização de *clusters* foi criada para aproveitar os melhores indivíduos da população inicial gerada. Este procedimento varre a população no sentido do melhor para o pior indivíduo gerando novos *clusters* ou assimilando os indivíduos em *clusters* já criados. Caso um indivíduo não se encontra dentro do raio r de nenhum dos *clusters* já existentes, novos *clusters* são criados. A assimilação de um indivíduo é feita no *cluster* cujo centro se encontra mais próximo ao indivíduo. A distância até o centro é medida como sendo o número de quantidade de trocas necessárias para um indivíduo se transformar no centro do *cluster*. O processo de inicialização de *clusters* termina quando toda a população é varrida ou quando é gerado o número máximo de *clusters*, C_{max} . Para determinar os valores de C_{max} , γ , α e r , foram realizados uma série de testes preliminares que serão explorados posteriormente.

Para assimilar um indivíduo a um *cluster*, utilizou-se como base o processo de *Path Relinking* (Glover, 1996). A partir de um indivíduo *ind* são realizadas sucessivas trocas de um par de genes até que o cromossomo se torne idêntico ao cromossomo do centro do *cluster*. A cada troca um novo cromossomo é gerado e avaliado e o par de genes que é escolhido para a troca é aquele que produzira o cromossomo com melhor avaliação a cada passo. Se no final, o próprio indivíduo assimilado *ind* ou um melhor cromossomo é encontrado nas sucessivas trocas tivesse uma melhor avaliação que o centro do *cluster*, este passaria a ser o novo centro do *cluster*. O processo evolutivo adotado teve como critério de parada o número de 50 tentativas de inserção de novos indivíduos a cada iteração. O processo de *Path Relinking* é descrito abaixo.

Passo 1: Seja $i = 1$, $count\ i = 0$ e o indivíduo a ser inserido p e o centro do *cluster* ind_c .

Passo 2: Se $i = n$, pule para o Passo 8, caso contrário, realize o Passo 3.

Passo 3: Se ind_i (posição da tarefa no indivíduo) e ind_{c_i} (posição da tarefa do indivíduo centro) forem iguais, faça Passo 7 caso contrário, faça Passo 4

Passo 4: Faça $j = i$.

Passo 5: Enquanto ind_{c_j} for diferente de ind_i faça $j = j + 1$.

Passo 6: Realize a troca das tarefas entre as posições i e j de ind_c e $count = count + 1$.

Passo 7: Faça $i = i + 1$ e faça Passo 2.

Passo 8: Retorne $count$ que é o valor da distância até o r do centro do *cluster*.

Na geração de um novo indivíduo, dois pais são selecionados na população: um entre os $\beta\%$ melhores da população, chamado de indivíduo *base*, e o outro entre todos da população, chamado de *guia*. O processo para gerar novos indivíduos utilizados foi o operador de cruzamento conhecido como *BOX (Block Order Crossover)* (Syswerda, 1989). Neste procedimento os indivíduos pais são combinados através da cópia aleatória de genes de ambos os pais, o que resulta na geração de um único filho contendo a carga genética dos pais. Cada filho gerado obtém 50% dos genes de cada pai. A Figura 5 ilustra o esquema de recombinação *BOX*.

Figura 5 - Recombinação BOX (Adaptado de Ribeiro Filho et al., 2007b)

Após a geração de um novo filho pelo processo de recombinação, o indivíduo passa por um processo de *Busca Local (VNS)* e em seguida com a probabilidade de $\alpha\%$ de passar por outro processo de busca local simples, denominado de *LSI*.

O novo indivíduo, após passar pelo processo de Busca Local, caso ele não seja idêntico a nenhum outro indivíduo pertencente à população, ele será inserido na população na posição referente à sua avaliação, fazendo assim, a eliminação do pior indivíduo presente na população até o momento, ou seja, o processo evolutivo atualiza a população a cada novo indivíduo gerado. Abaixo, segue o processo de inserção do indivíduo na população.

- Passo 1:** Caso não haja indivíduos na população, insira o primeiro indivíduo *ind* na população e faça Passo 4, caso contrário, faça Passo 2.
- Passo 2:** Insira o indivíduo na população de acordo com a sua qualidade.
- Passo 3:** Caso os vetores sejam iguais, descarte o indivíduo da população e faça Passo 5, caso contrário, Passo 4.
- Passo 4:** Inserção do indivíduo = *verdadeira*, sem seguida faça Passo 6.
- Passo 5:** Inserção do indivíduo = *falsa*.
- Passo 6:** Finalizado o processo de inserção do indivíduo na população.

Cada novo indivíduo inserido com sucesso na população, passa pelo componente *AI* da metaheurística e neste processo, o indivíduo é assimilado pelo *cluster* de cujo centro está mais próximo ou geram novos *clusters*. Durante a pesquisa, notou-se que nas primeiras iterações do processo evolutivo, o número de *clusters* aumentava, mas em seguida, declinava em função da eliminação dos *clusters* sem assimilações, aqueles referentes a regiões possivelmente não promissoras. Abaixo, segue o passo a passo do componente *AI*.

- Passo 1:** Através do processo de *Path Relinking*, calcule a distância entre os centros dos *clusters* existentes.
- Passo 2:** Caso a distância (número de movimentos para uma solução tornar-se ao centro) seja maior que r quando comparado com todos os raios dos *clusters* existentes, um novo *cluster* é criado e faça Passo 5, caso contrário, faça Passo 3.
- Passo 3:** Encontrado o *cluster* ao qual o indivíduo pertence, insira o indivíduo no *cluster*.
- Passo 4:** Se o indivíduo inserido apresenta qualidade superior ao centro do *cluster* inserido, o centro do *cluster* é atualizado, faça Passo 6.
- Passo 5:** Caso o número de *clusters* existentes seja maior que C_{max} , a inserção de *cluster* = *falsa* e faça Passo 7, caso contrário, faça Passo 6.
- Passo 6:** Inserção de *cluster* = *verdadeira*.
- Passo 7:** Finalizado o processo de Inserção de *cluster*.

Após o processo de geração de 50 novos indivíduos, cada um dos *clusters* é analisado pelo componente *AA* do método e nesta etapa, os *clusters* que passaram por alguma assimilação na iteração são considerados promissores. Os *clusters* promissores passam pelo

processo equivalente ao componente AO. Os *clusters* que não passaram por nenhuma assimilação, são eliminados. Abaixo, segue a sequência do AO.

Passo 1: $i = 1$.

Passo 2: Seja p_i o indivíduo centro do *cluster* i .

Passo 3: O indivíduo ind_i passará pelo processo LS2.

Passo 4: O indivíduo ind_i passará em seguida pelo processo de Busca Local VNS.

Passo 5: Insira o indivíduo ind_i na população, caso a inserção seja bem sucedida, faça Passo 6, caso contrário, faça Passo 7

Passo 6: Insira o indivíduo em um *cluster*, caso a inserção seja bem sucedida, remova o pior indivíduo.

Passo 7: Caso i seja menor que o número de *clusters*, faça $i = i + 1$ e faça Passo 2, caso contrário, faça Passo 8.

Passo 8: O melhor indivíduo da população será a resposta do algoritmo.

Durante o procedimento inteiro, o melhor *cluster* é aquele cujo centro tem a melhor avaliação e que é mantido salvo. No final das iterações, as soluções obtidas do problema são o primeiro indivíduo da população e o centro do melhor *cluster*, respectivamente gerados pelo componente *ME* (gerador de soluções) e pela evolução dos *clusters*.

A seguir, será apresentada em subitens a descrição de cada heurística utilizada no CS.

3.1.1. Nawaz-Enscore-Ham (NEH)

O algoritmo NEH é um método utilizado em vários trabalhos para diversos tipos de problemas e critérios devido a sua eficácia para a obtenção de soluções iniciais. Dada uma ordenação inicial das tarefas uma sequência inicial $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_k)$, o método constrói a solução final:

Passo 1: Considere $k = 2$.

Passo 2: Pegue a tarefa σ_k de σ e teste em todas as k posições possíveis de σ .

Passo 3: Insira a tarefa σ_k em σ na sequência parcial que resultar o menor valor total de atraso.

Passo 4: Faça $k = k + 1$, se $k = n$ Faça Passo 5, caso contrário faça Passo 2.

Passo 5: A sequência final σ' será o resultado do método.

3.1.2. Vizinhança de Inserção e Vizinhança de Permutação

Os algoritmos de *Insert Neighbourhood* (Vizinhança de Inserção – VI) e *Swap Neighbourhood* (Vizinhança de Permutação – VP) foram apresentados no trabalho de Ribas et al (2013) e são descritos a seguir.

3.1.2.1. Vizinhança de Permutação

A Vizinhança de Permutação consiste basicamente na permutação de duas tarefas de posições j e k em uma sequência onde a tarefa j é trocada de posição com a tarefa k . A vizinhança de permutação utilizada neste trabalho é definida como: para cada tarefa da sequência, vizinhanças são geradas com a troca de posição de uma tarefa com todas as outras posições das tarefas seguintes da sequência. Se a sequência (σ') é melhor que a atual sequência σ então a sequência (σ') torna-se a sequência atual e o processo continua até que todas as tarefas tenham sido consideradas. Para prevenir que todas as vizinhanças sejam exploradas na mesma sequência, as tarefas a serem permutadas são selecionadas de forma randômica. A seguir segue um passo a passo do método.

Passo 1: Seja $\sigma \{1, \dots, n\}$, a sequência das tarefas de entrada.

Passo 2: Selecione aleatoriamente uma tarefa $j \in \sigma$ entre as tarefas que ainda não foram selecionadas.

Passo 3: Permute de posição a tarefa j com todas as outras tarefas seguintes a esta, ou seja, para cada tarefa $k \in \sigma$, seguinte a tarefa j , troque de posição. Para cada troca, é gerada uma solução σ' .

Passo 4: Caso $TT(\sigma') < TT(\sigma)$, então $\sigma = \sigma'$.

Passo 5: Se todas as tarefas foram testadas, faça Passo 6, caso contrário, faça Passo 2.

Passo 6: A melhor solução encontrada será a solução final do procedimento.

3.1.2.2. Vizinhança de Inserção

A Vizinhança de inserção, em uma dada sequência, a tarefa de posição j é removida de sua posição e inserida na posição k . A vizinhança de inserção utilizada neste trabalho é

definida como: para cada tarefa da sequência, vizinhanças são geradas com a remoção de uma tarefa em uma sequência e inserida em todas as outras posições possíveis da sequência. Se a nova sequência gerada (σ') é melhor que a atual sequência (σ) então a sequência (σ') torna-se a sequência atual e o processo continua até que todas as tarefas tenham sido consideradas. Para prevenir que todas as vizinhanças sejam exploradas na mesma sequência, as tarefas a serem inseridas são selecionadas de forma randômica. A seguir segue um passo a passo do método.

Passo 1: Seja $\sigma \{1, \dots, n\}$, a sequência das tarefas de entrada.

Passo 2: Selecione aleatoriamente uma tarefa $j \in \sigma$ entre as tarefas que ainda não foram selecionadas.

Passo 3: Remova a tarefa j e insira em todas as outras posições da sequência σ . Para cada inserção, é gerada uma solução σ' .

Passo 4: Caso $TT(\sigma') < TT(\sigma)$, então $\sigma = \sigma'$.

Passo 5: Se todas as tarefas foram testadas, faça Passo 6, caso contrário, faça Passo 2.

Passo 6: A melhor solução encontrada será a solução final do procedimento.

3.1.3. Procedimento de Busca Local (VNS)

O procedimento de busca local utilizado é o apresentado no trabalho de Ribas et al (2013). Este procedimento resume-se basicamente em utilizar as vizinhança de permutação e vizinhança de inserção em conjunto. Segue abaixo o descritivo do método.

Passo 1: Seja $TT^* = TT(\sigma)$, $\sigma^* = \sigma$ e $nml1 = 0$.

Passo 2: Seja um número randômico $rand < \beta$ então $indmet = 0$ senão $indmet = 1$.

Passo 3: Faça $nml1 = nml1 + 1$ e $TT_o = TT(\sigma)$

Passo 4: Se $indmet = 0$ então realize Passo 5, caso contrário, realize o Passo 6.

Passo 5: Execute a vizinhança de permutação, após, faça Passo 7.

Passo 6: Execute a vizinhança de inserção., após, faça Passo 7.

Passo 7: Se $TT(\sigma) < TT_o$ ou $nml1 = 1$, execute o Passo 8, caso contrário, execute o Passo 9.

Passo 8: Faça $indmet = 1 - indmet$ e retorne ao Passo 3.

Passo 9: Solução final do processo.

Cada iteração do método utiliza ambos os métodos de busca de vizinhança, uma após a outra. A primeira vizinhança a ser explorada é selecionada de forma randômica com probabilidade de $\beta = 50\%$, representado no Passo 2. Após a exploração das vizinhanças, se a solução for melhorada, a solução atual é atualizada pela solução explorada, ou seja, a solução atual σ é comparada com a solução σ' explorada, caso a solução explorada é superior em qualidade, à solução atual é atualizada e o processo é continuado até que não haja mais melhoramento.

3.1.4. Procedimento LS1

O procedimento LS1, presente no CS, é um processo híbrido que utiliza os dois tipos de vizinhança (permutação e inserção, já exploradas anteriormente). A seguir, segue a metodologia do algoritmo LS1.

Passo 1: Seja σ a solução inicial.

Passo 2: $\sigma' = \text{vizinhança_inserção}(\sigma)$.

Passo 3: $\sigma'' = \text{vizinhança_permutação}(\sigma)$.

Passo 4: Se $TT(\sigma') < TT(\sigma)$ e $TT(\sigma') < TT(\sigma'')$ faça $\sigma = \sigma'$ e Passo 6, caso contrário, faça Passo 5.

Passo 5: Se $TT(\sigma'') < TT(\sigma)$ e $TT(\sigma'') < TT(\sigma')$ faça $\sigma = \sigma''$.

Passo 6: Retorna a sequência σ .

3.1.5. Procedimento LS2

O procedimento LS2, assim como o método LS1, é uma hibridização que utiliza os algoritmos de vizinhança de inserção e permutação. Segue abaixo, a metodologia do LS2.

Passo 1: Seja σ a solução inicial.

Passo 2: Caso *fim* seja verdadeiro ou o número *iter* de iterações seja maior que 5, faça Passo 10.

Passo 3: $iter = iter + 1$.

Passo 4: $\sigma' = \text{vizinhança_inserção}(\sigma)$.

Passo 5: Se $TT(\sigma') < TT(\sigma)$, faça $\sigma = \sigma'$, caso contrário, faça Passo 2.

Passo 6: $\sigma'' = \text{vizinhança_permutação}(\sigma)$.

Passo 7: Se $TT(\sigma'') < TT(\sigma)$, faça $\sigma = \sigma''$ e Passo 2, caso contrário, faça Passo 8.

Passo 8: $\sigma'' = \text{vizinhança_permutação}(\sigma)$.

Passo 9: Se $TT(\sigma'') < TT(\sigma)$, faça $\sigma = \sigma''$ e Passo 2, caso contrário, faça $\text{fim} = \text{verdadeiro}$ e Passo 2.

Passo 10: Retorne como solução a sequência σ .

Analisando o algoritmo, podemos verificar que são realizadas 5 iterações caso haja melhoria na sequência ao passar pelos algoritmos de vizinhança de inserção e permutação.

Caso haja melhoria no processo de inserção e nenhuma melhoria no processo de permutação em qualquer uma das iterações, o processo LS2 é encerrado e retorna a melhor sequência gerada pelo processo.

CAPITULO 4

4. *Iterated Local Search com Variable Neighbourhood Search*

A metaheurística *Iterated Local Search (ILS)* apresentada no trabalho de Ribas et al. (2013) é um método que aplica perturbações no processo de busca local com o objetivo de diversificar a busca. O algoritmo *ILS* é composto de quatro componentes: um procedimento que gera uma Solução Inicial, um mecanismo de perturbação que modifica uma solução σ em uma solução σ' , um procedimento de Busca Local e um *Acceptance Criterion* que decide se a sequência σ ou a sequência σ' será a sequência de entrada para o mecanismo de perturbação.

Abaixo, será apresentado o método *ILS*. Primeiro, uma solução inicial é gerada e o processo continua para a parte principal do algoritmo e esta parte é executada repetidamente até que o critério de parada utilizado seja atingido. No procedimento de busca local, cada iteração melhora a solução σ .

Passo 1: Seja σ a solução inicial.

Passo 2: Faça $\sigma^* = \sigma$ e $TT^* = TT(\sigma)$.

Passo 3: $\sigma' = \text{Busca Local}(\sigma)$.

Passo 4: Se $TT(\sigma') < TT^*$ faça $TT^* = TT(\sigma')$ e $\sigma^* = \sigma'$.

Passo 5: Se $TT^* < TT(\sigma')$ e $\text{rand} < \alpha$, faça $\sigma' = \sigma^*$.

Passo 6: $\sigma = \text{mecanismo de perturbação}(\sigma')$.

Passo 7: Se critério de parada atingido, finalize a metaheurística, caso contrário, execute o Passo 3.

4.1. Solução Inicial

Para se obter a solução inicial utilizada no método *ILS* para minimizar o tempo total de atraso para o problema *flow shop* com bloqueio, foi utilizada uma adaptação do Método NEH e como se sabe, o método possui duas etapas: primeiro, as tarefas são ordenadas seguindo a regra LPT e em após, passam por um processo iterativo de inserção. Ribas et al. (2013) apresentaram duas variantes de solução inicial para o método *ILS*, a primeira a *earliest due date* (EDD) e a *processing times and due dates* (FPD), algoritmo apresentado por Ronconi e Henriques (2009). A complexidade de ambas as variações de NEH é $O(n^3m)$. De acordo

com os estudos de Ribas et al. (2013), o método ILS apresentou melhor qualidade utilizando a sequência EDD como solução inicial para o método NEH.

4.2. Busca Local

A busca local utilizada para o método ILS, é a mesma utilizada neste trabalho para o *Cluster Search*, apresentada na seção 3.1.3.

4.3. Acceptance Criterion

O *acceptance criterion* evita que a solução se afaste muito da melhor solução encontrada. Se a solução atual σ' é pior do que a melhor solução encontrada σ^* , então a sequência atual σ' é definida como a melhor sequência encontrada σ^* com probabilidade $1 - \alpha$.

4.4. Mecanismo de Perturbação

No algoritmo ILS, o mecanismo de perturbação gera uma perturbação na solução atual que seja suficiente para a solução saia do mínimo local, mas não deve destruir completamente as características da solução obtida de modo a evitar um reinício aleatório. O método apresentado de perturbação apresenta três operadores de vizinhança: PI, EFSR e EBSR propostos por Della Groce, Narayan e Tadei (1996) para o problema de duas máquinas para o ambiente *flow shop* com o critério de tempo total.

- PI (*Pairwise Interchange*): dada uma sequência σ e duas posições $k1$ e $k2$, as tarefas são trocadas entre si, isto é, $\sigma = (5, 4, 1, 3, 2)$ e $k1 = 2$ e $k2 = 3$, resultado na sequência $\sigma' = (5, 1, 4, 3, 2)$.
- EFSR (*Extraction and Forward Shifted Re-insertion*): dada uma sequência σ e duas posições $k1$ e $k2$ com $k2$ na sequência posterior que $k1$, a tarefa $k1$ é extraída e reinserida na posição $k2$, isto é, $\sigma = (5, 4, 1, 3, 2)$ e $k1 = 1$ e $k2 = 4$, o resultado seria $\sigma' = (3, 5, 4, 1, 2)$
- EBSR (*Extraction and Backward Shifted Reinsertion*): dada a sequência σ e duas posições $k1$ e $k2$ com $k1$ antes de $k2$ na sequência, extraia a tarefa $k1$ e

reinsira na posição $k2$, isto é, $\sigma = (5, 4, 1, 3, 2)$ e $k1 = 1$ e $k2 = 4$, o resultado seria $\sigma' = (4, 1, 3, 5, 2)$.

A perturbação consiste em aplicar estes três operadores d vezes para duas posições $k1$ e $k2$ para dada uma sequência σ . A posição $k1$ é selecionada randomicamente entre todas as posições enquanto $k2$ é selecionado randomicamente somente entre os $2*d$ posições com maiores valores de atraso. Para cada tarefa, o seu valor de atraso é calculado e salvo em um vetor **lat**. As tarefas $2*d$ posições com os maiores valores de atraso são extraídas do vetor **lat** e colocadas em um vetor **font** que é usado para selecionar randomicamente d posições para posteriormente serem usadas nos três operadores de vizinhança. A principal ideia deste tipo de perturbação é tentar mover para frente algumas tarefas que estão atrasadas. Cada operador lida com uma nova sequência que é avaliada e a melhor é passada pelo processo de busca local VNS.

A seguir, segue um passo a passo do mecanismo de perturbação utilizado no método ILS.

Passo 1: Seja σ a solução inicial.

Passo 2: Preencha o vetor **lat** com os tempos de atraso de cada tarefa $i \in n$ da sequência inicial σ . $\text{Lat}[i] = \text{completion time}[i] - \text{due date}[i]$.

Passo 3: Selecione do vetor **lat** as $2*d$ posições com os maiores valores de atraso e coloque no vetor **font**. Elimine randomicamente d componentes de **font**. Ordene **font** em ordem crescente.

Passo 4: Faça $j = 1$.

Passo 5: $k2 = \text{font}(j)$.

Passo 6: Selecione randomicamente $k1$ entre as posições 1 e $k2-1$.

Passo 7: $i1 = \sigma(k1)$ e $i2 = \sigma(k2)$.

Passo 8: $\sigma1 = PI(\sigma, k1, k2)$.

Passo 9: $z1 = TT(\sigma1)$.

Passo 10: $EFSR(\sigma, i2, k1)$.

Passo 11: $z2 = TT(\sigma2)$.

Passo 12: $\sigma3 = EBSR(\sigma, i1, k2)$.

Passo 13: $z3 = TT(\sigma3)$

Passo 14: $z^* = \min\{\sigma1, \sigma2, \sigma3\}$: $\sigma^* = TT(z^*)$

Passo 15: $\sigma = \sigma^*$

Passo 16: Faça $j = j + 1$.

Passo 17: Caso $j \leq d$ faça o Passo 5. Caso contrário, faça Passo 18.

Passo 18: Retorne σ .

4.5. Ajuste dos parâmetros do algoritmo ILS

Para a pesquisa, foram utilizados os parâmetros ajustados no trabalho de Ribas et al. (2013), ou seja, a solução inicial utilizada e que apresentou melhores resultados foi a *earliest due date* (EDD) e os valores de d e α são, respectivamente, 4 e 0.75.

5. Experimentação Computacional

As estatísticas utilizadas para avaliar o desempenho dos métodos deste trabalho são: Índice de Desvio Relativo ou *relative deviation index* (RDI), conforme a expressão (7) que verifica o desvio da heurística H_{hs} em relação aos melhores e piores resultados obtidos na experimentação para um determinado problema, ou seja, quanto menor o valor, melhor a solução. O segundo critério é a Porcentagem de Sucesso, no qual ambos os métodos são avaliados em relação à quantidade de melhores soluções obtidas.

$$RDI = (H_{hs} - B_s) / (P_s - B_s) \quad (7)$$

Na equação (7), H_{hs} é a média dos valores obtidos pela heurística h em cinco rodadas na para o problema s , e B_s e P_s os melhores e piores valores obtidos para cada problema s considerando os resultados dos métodos a serem avaliados.

Os métodos descritos foram programados em linguagem de programação C em um microcomputador com sistema Windows e foi utilizado o compilador MinGW na plataforma IDE Eclipse. O microcomputador apresenta um processador Core i7 3.2GHz e 8GB de RAM. Para a experimentação, foi utilizado o banco de dados de Taillard (1993) que totaliza em 110 problemas com o número de máquinas $m \in \{5, 10, 20\}$ e o número de tarefas $n \in \{20, 50, 100, 200\}$ e a combinação $n \times m$ apresenta 10 problemas.

Os valores de *due date* foram distribuídos uniformemente entre $LB(1 - F - R/2)$ e $LB(1 - F + R/2)$ (Potts & Van Wassenhove, 1982) onde F é o fator de atraso das tarefas, R a faixa de dispersão, LB o *lower bound* que seria o limite inferior do *makespan* no ambiente *flowshop* de acordo com Taillard (1993).

Os valores de F e R utilizados para cada cenário foram:

- Cenário 1: baixo valor de atraso ($F=2$) e pequena faixa de datas de entrega ($R=0,6$).
- Cenário 2: baixo fator de atraso ($F=2$) e ampla faixa de datas de entrega ($R=1,2$).
- Cenário 3: alto fator de atraso ($F=0,4$) e pequena faixa de datas de entrega ($R=0,6$).

- Cenário 4: alto fator de atraso ($F=0,4$) e ampla faixa de datas de entrega ($R=1,2$).

Para efeitos comparativos, foram utilizadas as instâncias de Ronconi e Henriques (2009) que apresentam quatro cenários diferentes para cada problema e com isso, totalizando em uma amostragem de 440 problemas.

5.1. Ajustes experimentais dos parâmetros do método ECS

No método ECS, descrito anteriormente, há quatro parâmetros a serem avaliados: C_{max} (número máximo de *Clusters*), γ (Porcentagem de melhores indivíduos da População), α (Probabilidade de ocorrer a Busca Local) e r (Raio do *Cluster*). Para o valor de C_{max} , os testes mostraram que nas primeiras quatro ou cinco iterações do processo, o número de *clusters* aumentava, mas em seguida, declinava em função da eliminação dos *clusters* sem assimilações. Para escolher o valor, foram testados alguns valores para C_{max} com o critério de parada de $n^2m \cdot 10^{-5}$ s para a metaheurística. Na Tabela 2 apresentam-se os resultados em termos de RDI, para cada valor de C_{max} testados. Os valores testados foram: 50, 200 e 500.

Tabela 2: Testes para Valores de C_{max}

	C_{max}		
	50	200	500
20x5	0,396	0,321	0,459
20x10	0,445	0,368	0,197
20x20	0,377	0,359	0,303
50x5	0,541	0,426	0,302
50x10	0,357	0,459	0,592
50x20	0,56	0,484	0,443
100x5	0,393	0,344	0,495
100x10	0,41	0,521	0,472
100x20	0,434	0,528	0,526
200x10	0,485	0,422	0,441
200x20	0,588	0,494	0,473
Média	0,453	0,43	0,427

Na Tabela 2 é possível visualizar, que para os valores de C_{max} testados com o banco de dados para este tipo de problema, média do valor de RDI sofre pouca influência do limite estabelecido para a criação de *Clusters*. Com isso podemos concluir que os limites testados,

difficilmente são alcançados, isto é, o número máximo de grupos exerce pouca interferência na resposta do algoritmo. O valor escolhido para este trabalho foi 500.

Para o ajuste dos parâmetros r , α e γ , foi realizada uma bateria de testes, pois os parâmetros sofrem influência entre si dentro do algoritmo. Foram testados três níveis de valores para cada parâmetro, ou seja, resultando em um total de 27 condições diferentes. Os parâmetros podem possuir valores de 0 a 1, com isso, foram adotados os níveis 0,2, 0,5 e 0,8, ou seja, um valor próximo a 0, um valor mediano e um valor próximo a 1. A Tabela 3 apresenta as possíveis combinações entre os três parâmetros.

Tabela 3: Tabela de variações dos Parâmetros r , α e γ

<i>Condições</i>	γ	α	r
1	0,2	0,2	0,2
2	0,2	0,5	0,2
3	0,2	0,8	0,2
4	0,2	0,2	0,5
5	0,2	0,2	0,8
6	0,2	0,5	0,5
7	0,2	0,5	0,8
8	0,2	0,8	0,8
9	0,5	0,2	0,2
10	0,5	0,5	0,2
11	0,5	0,8	0,2
12	0,5	0,2	0,5
13	0,5	0,2	0,8
14	0,5	0,5	0,5
15	0,5	0,5	0,8
16	0,5	0,8	0,8
17	0,8	0,2	0,2
18	0,8	0,5	0,2
19	0,8	0,8	0,2
20	0,8	0,2	0,5
21	0,8	0,2	0,8
22	0,8	0,5	0,5
23	0,8	0,5	0,8
24	0,8	0,8	0,8
25	0,8	0,8	0,5
26	0,2	0,8	0,5
27	0,5	0,8	0,5

A Tabela 4 apresenta os valores de RDI da média de todos os problemas testados (440) para cada combinação dos parâmetros apresentado na Tabela 3. O critério de parada do algoritmo adotado foi o mesmo utilizado para os testes de valores para C_{max} .

Tabela 4: Resultado da Tabela 3

<i>Condições</i>	<i>RDI</i>
1	0,156
2	0,145
3	0,163
4	0,255
5	0,316
6	0,182
7	0,171
8	0,136
9	0,121
10	0,142
11	0,149
12	0,135
13	0,139
14	0,165
15	0,147
16	0,119
17	0,153
18	0,145
19	0,138
20	0,226
21	0,112
22	0,146
23	0,138
24	0,19
25	0,033
26	0,063
27	0,037

Através da Tabela 4 e da Figura 6 é possível concluir que a condição 25 apresentou o melhor resultado na média de todos os portes de problemas testado.

Figura 6: Gráfico das RDI das condições iniciais

A Tabela 5 apresenta os valores dos parâmetros que serão utilizados neste trabalho.

Tabela 5: Melhores Parâmetros

γ	α	r
0,8	0,8	0,5

5.2. Análise dos Resultados

Nesta seção, serão confrontados os resultados obtidos a partir do método ECS e o método ILS, o melhor método apresentado na literatura atual para o problema *Flow Shop* com bloqueio e *buffer zero* com a minimização do tempo total de atraso. O critério de parada da metaheurística utilizou-se o tempo computacional de $30 n^2 m 10^{-5}$ s para ambos os métodos.

A Tabela 6 apresenta as médias dos valores de RDI separados por cenário e por porte de problema da média dos valores da experimentação do método ECS. A Figura 7 auxilia a observação do comportamento dos valores de RDI de cada cenário por porte de problema. Como o esperado, os valores obtidos para problemas de pequeno porte (20 tarefas) apresentaram valores bem pequenos, próximos à zero, ou seja, os resultados obtidos em todas as rodadas foram semelhantes. Os valores obtidos de RDI para os cenários um, três e quatro, foram semelhantes considerando todos os portes de problema respectivamente, mas no cenário dois, os valores foram semelhantes aos demais cenários para portes de problemas com um número maior de máquinas. Nos demais portes, observam-se valores inferiores a 0,1 devido à presença maior de valores nulos de *total tardiness*.

Tabela 6: Valores de RDI do método ECS

	ECS			
	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	0,086	0,000	0,081	0,000
20x10	0,046	0,123	0,137	0,087
20x20	0,056	0,053	0,111	0,112
50x5	0,298	0,060	0,372	0,443
50x10	0,354	0,238	0,383	0,387
50x20	0,338	0,381	0,362	0,314
100x5	0,312	0,029	0,310	0,341
100x10	0,358	0,114	0,457	0,413
100x20	0,473	0,328	0,436	0,419
200x10	0,309	0,025	0,276	0,375
200x20	0,305	0,335	0,348	0,311
Média	0,267	0,153	0,298	0,291

A Figura 8 ilustra a média do valor de RDI por cenário. Neste gráfico é possível confirmar uma tendência semelhante entre os cenários um, dois e quatro. Como observado anteriormente, o valor de RDI do cenário dois, na média, é inferior aos demais devido à característica deste cenário de ter como resultado uma grande quantidade de valores nulos.

Figura 7: Gráfico dos valores de RDI por porte de problema do método ECS

Figura 8: Gráfico da média dos valores de RDI de cada Cenário utilizando o método ECS

A Tabela 7 apresenta as médias dos valores de RDI obtidas com o método ILS e separadas por cenário e por porte de problema da média dos valores da experimentação. A Figura 9 auxilia a observação do comportamento dos valores de RDI de cada cenário por porte de problema. Os valores obtidos para os cenários um, três e quatro, apresentaram um comportamento semelhante de valores de RDI em todos os portes de problemas respectivamente, mas no cenário dois, o comportamento entre os portes de problemas foram semelhantes aos apresentados na Figura 7, ou seja, um comportamento característico do cenário. A Figura 10 apresenta a média dos valores de RDI por cenário, semelhante em comportamento ao gráfico da Figura 8 ao longo dos cenários.

Tabela 7: Valores de RDI do método ILS

	ILS			
	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	0,415	0,020	0,422	0,181
20x10	0,410	0,464	0,321	0,394
20x20	0,405	0,130	0,491	0,464
50x5	0,529	0,105	0,502	0,496
50x10	0,546	0,490	0,553	0,599
50x20	0,500	0,500	0,549	0,499
100x5	0,606	0,061	0,533	0,574
100x10	0,576	0,262	0,529	0,553
100x20	0,634	0,583	0,671	0,560
200x10	0,666	0,147	0,633	0,634
200x20	0,646	0,594	0,627	0,624
Média	0,539	0,305	0,530	0,507

Figura 9: Gráfico dos valores de RDI por porte de problema do método ILS

Figura 10: Gráfico da média dos valores de RDI de cada Cenário do método ILS

As Figura 11 e Figura 12 apresentam gráficos comparativos entre os métodos ECS e ILS separados por portes de problema e por cenários. É possível observar na Figura 11 que o

método ECS foi superior ao método ILS em todos os portes de problemas quando se consideram os valores de RDI resultantes. Como consequência, a Figura 12 exibe a superioridade do método quando se considera a média em todos os cenários.

Figura 11: Gráfico comparativo das Médias de RDI por porte de problema dos métodos ECS e ILS

Figura 12: Gráfico comparativo das Médias de RDI por cenário dos métodos ECS e ILS

A Tabela 8 apresenta os valores referentes à porcentagem de sucesso para o método ECS do melhor valor entre as cinco rodadas da experimentação. Com o auxílio da Figura 13 é possível notar que a porcentagem de sucesso obtida nas experimentações, na média, ficou em torno de 80%, ou seja, o método obteve na experimentação resultados iguais ou superiores ao método ILS na maioria das tarefas testadas.

Tabela 8: Porcentagem de Sucesso do método ECS para os melhores resultados

	ECS			
	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	100	100	100	100
20x10	100	100	100	100
20x20	100	90	100	100
50x5	70	100	40	30
50x10	70	90	70	90
50x20	70	50	50	80
100x5	80	100	90	90
100x10	80	80	40	70
100x20	80	70	90	80
200x10	90	100	100	90
200x20	90	70	90	90
Média	84,545	86,364	79,091	83,636

Figura 13: Gráfico da Porcentagem de Sucesso do melhor resultado do Método ECS referente a Tabela 8

A Tabela 9 apresenta os valores da porcentagem de sucesso dos melhores resultados obtida na experimentação para o método ILS. Através da Figura 14, somente o cenário 2 apresentou um valor de porcentagem de sucesso expressivo (devido à característica do cenário), com isso, podemos concluir que os melhores resultados obtidos neste método foram inferiores na maioria dos problemas.

Tabela 9: Porcentagem de Sucesso do método ILS para os melhores resultados

	ILS			
	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	80	100	80	100
20x10	90	70	100	80
20x20	80	100	70	70
50x5	30	90	60	70
50x10	30	30	30	10
50x20	30	50	50	20
100x5	20	100	10	10
100x10	20	70	60	30
100x20	20	30	10	20
200x10	10	80	0	10
200x20	10	30	10	10
Média	38,182	68,182	43,636	39,091

Figura 14: Gráfico da Porcentagem de Sucesso do melhor resultado do Método ILS referente a Tabela 9

A Figura 15 apresenta a comparação entre as médias das porcentagens de sucesso dos melhores valores obtidos da experimentação para os métodos ECS e ILS divididas por cenário. Os valores obtidos no método ECS apresentaram grande superioridade na maioria dos cenários quando comparados ao método ILS, apenas no cenário 2 a porcentagem de sucesso foi semelhante, ou seja, neste caso ambos os métodos apresentaram valores iguais entre si e isso se deve a característica do cenário que apresenta diversos valores nulos de *total tardiness*.

Figura 15: Gráfico do confronto entre as médias das porcentagens de sucesso entre os melhores resultados

A Tabela 10 apresenta os valores da porcentagem de sucesso da média dos valores de *total tardiness* obtidos na experimentação do método ECS. Com o auxílio da Figura 16, podemos observar que a média dos valores, por cenário, apresentou um sucesso em torno de 90%, ou seja, os valores obtidos pelo método ECS para os problemas testados foram superiores ao método ILS em sua grande maioria.

Tabela 10: Porcentagem de Sucesso do método ECS para a média dos resultados

	ECS			
	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	100	100	100	100
20x10	100	90	90	100
20x20	100	90	100	100
50x5	100	100	80	70
50x10	80	100	100	80
50x20	80	80	90	80
100x5	90	100	90	90
100x10	90	100	70	70
100x20	60	80	100	90
200x10	100	100	100	90
200x20	100	100	100	100
Média	90,909	94,545	92,727	88,182

Figura 16: Gráfico da Porcentagem de Sucesso da média dos resultados do Método ECS referente a Tabela 10

A Tabela 11 exibe a porcentagem de sucesso da média dos valores obtidos na experimentação para o método ILS. A Figura 17 ilustra o comportamento do método ILS por porte de problema. Analisando os dados, na média por cenário, a Porcentagem de Sucesso apresentou valores em torno de 20%, isto é, os valores médios de *total tardinees* obtidos na experimentação para o método ILS apresentaram uma inferioridade significativa quando

confrontados aos valores obtidos com o método ECS. Como observado anteriormente, o cenário 2 apresentou uma porcentagem de sucesso maior aos outros cenários devido à sua característica de apresentar valores de *total tardines* nulo.

Tabela 11: Porcentagem de Sucesso do método ILS para a média dos resultados

	ILS			
	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	20	90	20	60
20x10	20	40	30	30
20x20	20	50	10	10
50x5	0	70	20	30
50x10	20	0	0	20
50x20	20	20	10	20
100x5	10	90	10	10
100x10	10	40	30	30
100x20	40	20	0	10
200x10	0	50	0	10
200x20	0	0	0	0
Média	14,545	42,727	11,818	20,909

Figura 17: Gráfico da Porcentagem de Sucesso do Método ILS referente a Tabela 11

A Figura 18 confronta os valores médios da Tabela 10 e Tabela 11. Agrupados por cenário, podemos observar que neste critério, em todos os cenários, o método ECS foi superior ao método ILS com grande vantagem. O método ECS obteve percentagens em torno de 90%, ou seja, o método forneceu quase em sua totalidade os melhores resultados.

Figura 18: Gráfico do confronto entre as médias das percentagens de sucesso entre a média dos resultados

A Tabela 12 apresenta os valores médios de RDI por porte de problema da média dos valores obtidos na experimentação. O gráfico da Figura 19 ilustra o confronto e a superioridade do método ECS sobre o método ILS em todos os portes de problema. O método ECS apresentou valores inferiores a 0,4 em quase todos os portes enquanto o método ILS apresentou valores superiores a 0,4 em quase todos os portes de problemas. Nesta comparação é possível analisar de uma maneira macro a superioridade do método ECS.

Tabela 12: Valores da média de RDI dos métodos ECS e ILS por porte de problema

	ECS	ILS
20x5	0,042	0,259
20x10	0,098	0,397
20x20	0,083	0,372
50x5	0,293	0,408
50x10	0,340	0,547
50x20	0,349	0,512
100x5	0,248	0,443
100x10	0,336	0,480
100x20	0,414	0,612
200x10	0,246	0,520
200x20	0,325	0,623
Média	0,252	0,470

Figura 19: Gráfico da média dos valores de RDI por porte de problema

Tabela 13: Valores de *Best-Know* atualizada para o critério de *Total Tardiness*

	Cenário 1	Cenário 2	Cenário 3	Cenário 4
20x5	627	897	3630	778
	659	0	1245	5140
	626	623	3183	3689
	1116	0	3323	1144
	454	15	2243	4045
	946	406	3395	1848
	777	518	2132	1821
	567	233	3305	4774
	700	571	2566	4086
	840	1303	3163	3402
20x10	1580	3136	6713	5865
	3356	909	6813	4130
	2549	5405	5661	3764
	1844	3248	6180	5620
	1988	4278	5728	7032
	2484	3034	5153	5123
	1448	255	3970	5850
	2694	3905	6577	6571
	2778	401	7735	7363
	4033	5214	7572	6317
20x20	6024	11679	11408	10039
	6871	6173	15373	20075
	7714	8889	14560	13283
	6014	10641	12674	15035
	7604	8735	13313	14394
	6625	6256	13912	15788
	6544	5893	11777	12654
	4391	5791	11429	9450
	8486	7937	12307	12831
	6502	4819	10896	15092
50x5	860	0	10431	1920
	2568	0	10426	9524
	2091	39	10224	10139
	1713	0	11396	11549
	1276	0	9908	8375
	1652	469	16457	9284
	981	0	9420	16078
	2732	0	13644	11235
	1926	0	11170	7656
	3502	0	9519	12197
50x10	7473	2803	24604	23813
	5781	182	26110	18008
	7214	1578	18631	22307
	6764	380	18442	22831

50x20	10956	4768	19886	25426
	4965	978	22928	21746
	3552	540	17076	19513
	5053	1956	19692	15881
	7519	3212	24186	20559
	4921	320	22543	16392
	12061	9909	45303	50137
	13214	9929	38002	40208
	9241	14412	36512	44537
	14524	9855	36570	51359
	16562	3133	36399	33767
	12437	18520	33093	35040
	10602	10602	36286	44091
	13741	20652	3829	36845
	12717	10846	36573	16572
	14355	7419	33702	40684
100x5	3468	0	27248	29511
	4973	0	42707	13741
	3808	0	43732	13149
	3811	450	32496	35539
	4158	0	36210	19681
	4738	0	35691	33990
	6608	0	36686	17359
	9952	0	38798	21325
	2911	0	41883	21306
	6208	0	42449	50337
100x10	12393	6322	61310	46735
	17227	1413	64298	48788
	16722	0	65259	48843
	18089	0	73308	61722
	24910	426	69075	56532
	20776	0	66231	24336
	16117	0	75731	46514
	12677	0	70110	69651
	14720	0	55469	37554
	9321	10613	61978	37999
100x20	37300	22563	115488	118967
	34960	0	102584	87685
	33222	3804	104583	98128
	30204	7774	101050	131060
	35395	8193	107382	89851
	32997	17452	101036	90324
	30839	16019	106958	113297
	36306	3249	112849	105291
	33883	1406	108414	96230
	22810	5574	99876	98274

200x10	48815	0	243735	165219
	67431	0	238313	203553
	52372	0	227424	233973
	46632	0	245231	185865
	77534	0	258312	179114
	78270	0	238429	192556
	49571	0	279334	169891
	61445	0	262739	152842
	68585	0	249239	210446
	57968	0	218831	279907
200x20	115388	25547	356598	254706
	130706	43714	391679	299680
	148546	19880	333927	282631
	129288	8084	390543	310403
	121748	922	312305	303571
	117989	11626	361329	442148
	109636	27171	367324	269119
	122667	29725	360429	259775
	137164	11076	356784	249295
	127646	1419	368280	350078

A Tabela 13 apresenta os valores atualizados de *best-know* de Ribas et al. (2013). Estes valores foram obtidos durante a pesquisa realizada neste trabalho na solução do problema *Flow Shop* com bloqueio com a minimização total do atraso. Esta atualização dos valores de *best-know* contempla no total de 67 novos valores destacados em negrito.

6. Considerações Finais

Neste trabalho foi abordado o problema *flow shop* com bloqueio com o critério de minimização do tempo total de atraso. O principal objetivo da pesquisa foi o desenvolvimento e a aplicação de um método metaheurístico que resolvesse o problema com um desempenho superior ao método da literatura atual que apresentava as melhores soluções. Para o confronto entre o método proposto e o método da literatura, foram utilizados os critérios de desempenho RDI (*relative deviation index*) e Porcentagem de Sucesso. Para cada critério, foram realizadas análises em diversas perspectivas de estratificação (por cenário, porte de problema e critério de avaliação) dos resultados para verificar se o desempenho do método proposto apresentou melhoria somente em uma combinação específica ou somente em uma visão geral.

A metaheurística desenvolvida neste trabalho baseia-se no método *Clustering Search* apresentada primeiramente por Oliveira e Lorena (2004, 2007). Para a solução do problema *flow shop* com bloqueio, foram realizadas adaptações nas etapas do algoritmo visando obter melhores soluções para o critério do tempo total de atraso.

O Método CS baseia-se no agrupamento de soluções para orientar a busca em regiões promissoras. Utilizando o sistema de agrupamento com a sistemática de busca local do método VNS, foram obtidos resultados promissores.

A pesquisa contemplou uma extensa experimentação computacional nos quais foram adquiridos resultados de cinco execuções de cada método utilizando o mesmo critério de parada do algoritmo. Foi realizada a análise com os valores médios obtidos na experimentação e com o melhor valor obtido durante a experimentação.

Na primeira etapa da experimentação, foram analisados os valores RDI da média dos valores obtidos na experimentação. Quando ambos os métodos são confrontados, o método ECS apresentou superioridade em todos os portes de problemas e em todos os cenários quando confrontado com o método ILS.

Na segunda etapa, foi realizada a análise da porcentagem de sucesso para os melhores valores de *total tardiness* obtidos na experimentação de cada método. Em mais de 80% dos problemas, o método ECS apresentou resultado igual ou superior ao método ILS. A partir desta análise, é possível concluir que o método ECS forneceu, em quase a totalidade dos casos, as melhores soluções da experimentação.

Em uma terceira etapa, realizou-se a análise da porcentagem de sucesso da média dos valores obtidos na experimentação. Este critério consiste em se analisar se há uma repetitividade consistente do método em prover boas soluções. Nesta análise, na média de cada tarefa, o método ECS foi igual ou superior ao método ILS em mais de 90% das tarefas analisadas.

Em uma última análise, os valores médios de RDI dos valores médios da experimentação foram agrupados por porte de problema. Através dos dados obtidos e dos gráficos ilustrativos, é possível notar que o método ECS foi superior ao método ILS em todos os portes de problemas analisado neste critério.

Após todas as análises realizadas com o método *Clustering Search* com o Método ILS, o método apresentado neste trabalho apresentou desempenho superior nos critérios medidos, o RDI e a Porcentagem de Sucesso.

Durante toda a experimentação computacional realizada na pesquisa, foi possível atualizar a tabela de *benchmark* dos 440 problemas analisados. No total, foram atualizados 67 novos valores de *total tardiness* e esta nova tabela servirá de referência para trabalhos futuros referentes ao problema *flow shop* com bloqueio e *buffer zero* para a minimização do tempo total de atraso.

6.1.Sugestões para trabalhos futuros

Após a análise do *Clustering Search* utilizado para a solução do problema *flow shop* com bloqueio com *buffer zero* para a minimização do atraso total da produção, foi possível considerar alguns pontos para pesquisas futuras:

- Neste trabalho foi utilizado o algoritmo genético na geração de novas soluções. Há a possibilidade de estudo de um novo algoritmo para a geração de novas soluções.
- Caso sejam desenvolvidas novas soluções de busca local em pesquisas futuras, há a possibilidade de se aplicar estas novas metodologias de busca local para obter um desempenho melhor do algoritmo *Clustering Search*.
- O algoritmo adaptado do *Clustering Search* aplicado neste trabalho pode ser utilizado em diversos tipos de problema *flow shop*, com a possibilidade de gerar soluções ótimas.
- Alterando o critério de parada do algoritmo (aumentando o tempo computacional de parada), o algoritmo genético presente tem a capacidade de

gerar diversas soluções diferentes que com o decorrer da metaheurística, ela agrupará estas soluções geradas, aumentando as possibilidades de surgirem soluções melhores.

- A tabela atualizada do *benchmark* dos valores do tempo total de atraso servirá de comparação para futuros trabalhos na análise de desempenho de métodos futuros.

Referências bibliográficas

- ABADI, I.N.K.; HALL, N.G.; SRISKANDARAJH, C. (2000). Minimizing cycle time in a blocking flowshop. *Operations Research*, v.48, p.177–180.
- ARMENTANO VA, RONCONI DP, (2000) Minimização do Tempo Total de Atraso no Problema FlowShop com *Buffer* Zero Através de Busca Tabu. *Gestão & Produção*, v 7, p353-63.
- BAGCHI, T.P.; GUPTA, J.N.D.; SRISKANDARAJAH, C. (2006). A review of TSP based approaches for flowshop scheduling. *European Journal of Operational Research*, v.169, p.816-854.
- BAUTISTA, J.; CANO, A.; COMPANYS, R.; RIBAS, I. (2012). Solving the Fm/block/Cmax problem using Bounded Dynamic Programming. *Engineering Applications of Artificial Intelligence*, v.25, p.1235–1245.
- CARAFFA, V.; IANES, S.; BAGCHI, T.P.; SRISKANDARAJAH, C. (2001). Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics*, v.70, p.101–115.
- COMPANYS, R.; MATEO, M. (2007). Different behaviour of a double branch-and-bound algorithm on Fm|prmu|Cmax and Fm|block|Cmax problems. *Computers & Operations Research*, v.34, p.938-953.
- COMPANYS, R.; RIBAS, I. (2014), A New Constructive Heuristic for the Fm|Block| $\sum T$, *7th International Conference on Industrial Engineering and Industrial Management - XVII Congreso de Ingeniería de Organización*.
- DAVENDRA, D.; BIALIC-DAVENDRA, M. (2013). Scheduling flow shops with blocking using a discrete self-organising migrating algorithm. *International Journal of Production Research*, v.51, p.2200-2218.
- GEEM, Z.W.; KIM, J.H.; LOGANATHAN, G.V. (2001). A new heuristic optimization algorithm: Harmony search. *Simulations*, v.76, p.60–68.
- GLOVER, F, 1996. Tabu search and adaptive memory programing: Advances, applications and challenges. *Interfaces Comput. Sci. Oper. Res.*, 1–75.
- GONG, H.; TANG, L.; DUIN, C.W. (2010). A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers & Operations Research*, v.37, p.960-969.

- GRAHAM, R. L., E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOY KAN. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics* 5: 287-326.
- GRABOWSKI, J.; PEMPERA, J. (2007). The permutation flow shop problem with blocking. A tabu search approach. *Omega*, v.35, p.302–311.
- GRABOWSKI, J.; SKUBALSKA, E.; SMUTNICKI, C. (1983). On flow shop scheduling with release and due dates to minimize maximum lateness. *Journal of Operational Research Society*, v.34, p.615–620.
- GUANLONG, D.; ZHENHAO, X.; XINGSHENG, G. (2012). A Discrete Artificial Bee Colony Algorithm for Minimizing the Total Flow Time in the Blocking Flow Shop Scheduling. *Chinese Journal of Chemical Engineering*, v.20, p.1067-1073.
- HALL, N.G.; SRISKANDARAJAH, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, v.44, p.510–525.
- HAN, Y.Y.; PAN, Q.K.; LI, J.Q.; SANG, H.Y. (2012a). An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *International Journal Advanced Manufacturing Technology*, v.60, p.1149–1159.
- HAN, Y.Y.; LIANG, J.J.; PAN, Q.K.; LI, J.Q.; SANG, H.Y. CAO, N.N. (2013). Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. *Journal Advanced Manufacturing Technology*, v.67, p.397-414.
- HAN, Y.Y.; DUAN, J.H.; YANG, Y.J.; ZHANG, M.; YUN, B. (2012b). Minimizing the Total Flowtime Flowshop with Blocking Using a Discrete Artificial Bee Colony. *Lecture Notes in Computer Science*, v.6839, p.91-97.
- HAN, Y. Y.; GONG, DUNWEI; SUN, XIAOYAN, (2014). A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. *Engineering Optimization*, 2014
- JOHNSON, L.A.; MONTGOMERY, D.C. (1974). Operations Research in Production Planning, Scheduling and Inventory Control. New York: Wiley.
- KOMESU, A. S. (2012). Avaliação de Métodos Heurísticos para minimização do *makespan* em ambiente *flowshop* com bloqueio. Monografia - Escola de Engenharia de São Carlos da Universidade de São Paulo.
- LEISTEN, R. (1990). Flowshop sequencing problems with limited *buffer* storage. *International Journal of Production Research*, v.28, p.2085–2100.
- LIN, S.W.; YING, K.C. (2013). Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. *Omega*, v.41, p.383–389.

- LIU, S.Q.; KOZAN, E. (2009). Scheduling a flow shop with combined *buffer* conditions. *International Journal Production Economics*, v.117, p.371–380
- MACCARTHY, B.L.; LIU, J.Y. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, v.31, p.59-79.
- MARTINEZ, S.; DAUZÈRE-PÉRÈS, S.; GUÉRET, C.; MATI, Y.; SAUER, H. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, v.169, p.855–864.
- MCCORMICK, S.T.; PINEDO, M.L.; SHENKER, S.; WOLF, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, v.37, p.925–936.
- MOSLEHI, G.; KHORASANIAN, D. (2013). Optimizing blocking flow shop scheduling problem with total completion time criterion. *Computers & Operations Research*, v.40, p.1874-1883.
- NAWAZ, M.; ENSCORE, E.E.; HAM, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, v.11, p.91–95.
- NAGANO, M.S.; DA SILVA, A.A.; LORENA, L.A.N. (2012). A new evolutionary clustering search for a no-wait flow shop problem with set-up times. *Engineering Applications of Artificial Intelligence*, v.25, p.1114-1120.
- NOWICKI, E. (1999). The permutation flow shop with *buffers*: A tabu search approach. *European Journal of Operational Research*, v.116, p.205–219.
- OLIVEIRA, A.C.M.; LORENA, L.A.N. (2004). Detecting promising areas by evolutionary clustering search. In: Advances in Artificial Intelligence, Bazzan, A.L.C. and Labidi, S. (eds) Springer *Lecture Notes in Artificial Intelligence*, p.385–394.
- OLIVEIRA, A.C.M.; LORENA, L.A.N. (2007). Hybrid evolutionary algorithms and clustering search. In: Grosan, C.; Abraham, A.; Ishibuchi, H. (eds). *Hybrid Evolutionary Systems - Studies in Computational Intelligence - Springer SCI Series*, v.75, p.81–102.
- PAN, Q.K.; WANG, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, v.40, p. 218–229.
- PAN, Q.K.; WANG, L.; GAO L. (2011b). A chaotic harmony search algorithm for the flow shop scheduling problem with limited *buffers*. *Applied Soft Computing*, v.11, p.5270–5280.
- PAN, Q.K.; WANG, L.; GAO, L.; LI, W.D. (2011a). An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate *buffers*. *Information Sciences*, v.181, p.668–685.

- PAN, Q.K.; WANG, L.; SANG, H.Y.; LI, J.Q.; LIU, M. (2013). A high performing memetic algorithm for the flowshop scheduling problem with blocking. *IEEE Transactions on Automation Science and Engineering*, v.10, p.741-756.
- PAPADIMITRIOU, C.; KANELLAKIS, P. (1980). Flow-shop scheduling with limited temporary storage. *Journal of the Association for Computing Machinery*, v.27, p.533-549.
- POTS, C.N.; VAN WASSENHOVE, L. N. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters* 1982;1:177-81
- QIAN, B; WANG, L.; HUANG, D.X.; WANG, X. (2009). An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. *International Journal of Production Research*, v.36, p.209-233.
- REDDI, S.S.; RAMAMOORTHY, C.V. (1972). On the flowshop sequencing problem with no-wait in process. *Operational Research Quarterly*, v.23, p.323-330.
- RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. (2011). An iterated greedy algorithm for the flowshop scheduling with blocking. *Omega*, v.39, p.293-301.
- RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. (2013). An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. *International Journal of Production Research*, DOI:10.1080/00207543.2013.802390.
- RIBEIRO FILHO, G.; NAGANO, M.S.; LORENA, L.A.N. (2007a). Hybrid Evolutionary Algorithm for Flowtime Minimisation in No-Wait Flowshop Scheduling. A.Gelbukh and A.F.Kuri Morales (Eds.): *Lecture Notes in Artificial Intelligence*, v.4827, p. 1099-1109.
- RIBEIRO FILHO, G.; NAGANO, M.S.; LORENA, L.A.N. (2007b). Evolutionary Clustering Search for Flowtime Minimization in Permutation Flow Shop. *Lecture Notes in Computer Science. Lecture Notes in Computer Science*, v.4771, 69-8.
- RONCONI, D.P.; ARMENTANO, V.A. (2001). Lower bounding schemes for flowshops with blocking in-process. *Journal of the Operational Research Society*, v.52, p.1289-1297.
- RONCONI, D.P. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, v.87, p.39-48.
- RONCONI, D.P. (2005). A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. *Annals of Operations Research*, v.138, p.53-65.
- RONCONI, D.P.; HENRIQUES, L. R. S. (2009). Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. *Omega*, v.37, p.272-281.
- SYSWERDA, G. 1989. Uniform crossover in genetic algorithms. In: *International Conference on Genetic Algorithms (ICGA)*. Virginia, USA, pp. 2-9.

- TAILLARD, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, v.64, p.278–285.
- Tasgetiren, M.F.; Pan, Q.K.; Suganthan, P.N.; Chen, A.H.L. (2011). A discrete artificial bee colony algorithm for the total flow time minimization in permutation flow shops. *Information Sciences*, v.181, p3459-3475.
- TRABELSI, W.; SAUVEY, C.; SAUER, N. (2012). Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. *Computers & Operations Research*, v.39 p.2520–2527.
- WANG, L.; PAN, Q.K.; TASGETIREN, M.F. (2010b). Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Systems with Applications*, v.37, p.7929–7936.
- WANG, L.; PAN, Q.K.; TASGETIREN, M.F. (2011). A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. *Computers & Industrial Engineering*, v.61, p.76–83.
- WANG, L.; PAN, Q.K.; SUGANTHAN, P.N.; WANG, W.H.; WANG, Y.M. (2010a). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*, v.37, p.509–520.
- WANG, X.; TANG, L. (2012). A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. *Applied Soft Computing*, v.12, p.652–662.
- WANG, C.; SONG, S.; GUPTA, J.N.D., WU, C. (2012). A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. *Computers & Operations Research*, v.39, p.2880–2887.
- XIE, Z.; ZHANG, C.; SHAO, X.; YIN, Y. (2014). Minimizing total flow time in a flow shop with blocking using a hybrid variable neighborhood search and simulated annealing algorithm. *Applied Mechanics and Materials Vols. 631-632 (2014) pp 57-61*.