UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DE SAÚDE

DEPARTAMENTO DE COMPUTAÇÃO

WAGNER AMORIM DA SILVA ALTOÉ

UMA META-HEURÍSTICA *CLUSTERING SEARCH* APLICADA AO PROBLEMA *JOB-SHOP SCHEDULING* FLEXÍVEL BI-OBJETIVO

ALEGRE

2017

WAGNER AMORIM DA SILVA ALTOÉ

UMA META-HEURÍSTICA *CLUSTERING SEARCH* APLICADA AO PROBLEMA *JOB-SHOP SCHEDULING* FLEXÍVEL BI-OBJETIVO

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e de Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. M.Sc. Dayan de Castro Bissoli.

ALEGRE

2017

WAGNER AMORIM DA SILVA ALTOÉ

UMA META-HEURÍSTICA *CLUSTERING SEARCH* APLICADA AO PROBLEMA *JOB-SHOP SCHEDULING* FLEXÍVEL BI-OBJETIVO

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em ____ de julho de 2017.

COMISSÃO EXAMIDADORA

_____

Prof. M.Sc. Dayan de Castro Bissoli

Universidade Federal do Espírito Santo

Orientador

_____

Prof. Dr. Geraldo Regis Mauri

Universidade Federal do Espírito Santo

_____

Prof. M.Sc. Flávio Tongo da Silva

Instituto Federal do Espírito Santo

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois tudo que tenho na vida foi conquistado pela tua bênção.

À minha família, pelo incentivo e apoio para continuar estudando todos esses anos, que não mediram esforços nenhum para me ajudar em qualquer problema.

À minha irmã, por entender que eu a amo, mesmo estando sempre ausente.

À minha namorada Larissa Brito, que esteve sempre me apoiando em todos esses anos, não me deixar desistir.

A todos os amigos que eu conheci nessa faculdade, que sempre estavam dispostos a ajudar, mesmo quando as vezes parecia impossível.

Ao meu orientador Dayan de Castro Bissoli, pela disponibilidade em ajudar, pela paciência e seus conhecimentos repassados durante todo o desenvolvimento do trabalho.

À Universidade Federal do Espírito Santo e todos os seus funcionários, principalmente os professores, que puderam transmitir seus conhecimentos de forma empolgante, facilitando mais ainda o aprendizado.

A empresa júnior Click Júnior por me proporcionar uma pequena experiência do mercado de trabalho aumentando assim o conhecimento adquirido nas aulas.

A todos, muito obrigado.

RESUMO

A programação de produção é um dos mais importantes processos de tomada de decisão nas indústrias. Nesse contexto, encontra-se o problema, conhecido como *Job Shop Scheduling* (JSP), que visa otimizar a alocação de recursos de produção em centros de trabalho. O problema *Job Shop* Flexível (FJSP) é um caso específico do JSP, que se mostra mais perto da realidade, possibilitando que uma operação de um determinado *job* seja realizado por mais de uma máquina. Considerando que para algumas indústrias é relevante se importar com mais de um objetivo, possibilitando assim uma melhor tomada decisão, o FJSP é tratado de forma multiobjetiva. Com isso, torna-se interessante produzir um conjunto de soluções distintas potencialmente boas, deixando ao encargo do tomador de decisão escolher qual solução será mais importante para a empresa. Considerando a complexidade do problema, é proposta a meta-heurística *Clustering Search* para solucioná-lo.

Palavras-chave: Clustering Search; makespan; atraso total; multiobjetivo; *Job Shop* Flexível;

Absctract

Production scheduling is one of the most important decision-making processes in industries. In this context, the problem is known as Job Shop Scheduling (JSP), which aims to optimize the allocation of production resources in work centers. The Flexible Job Shop (FJSP) problem is a specific case of the JSP, which is closer to reality, allowing an operation of a given job to be performed by more than one machine. Considering that for some industries it is important to care about more than one objective, thus enabling a better decision making, the FJSP is treated in a multiobjective way. Thus, it is interesting to produce a set of potentially good different solutions, leaving it to the decision-maker to choose which solution will be most important for the company. Considering the complexity of the problem, the metaheuristic Clustering Search is proposed to solve it.

Keywords: Clustering Search; makespan; tardiness; *multiobjective; Flexible Job Shop.*

LISTA DE FIGURAS

Figura 1 – Nova estrutura da instância.....24

Figura 2 – Representação colorida de parte da instância Mk01.....25

Figura 3 – Exemplo da representação de uma solução 26

Figura 4 - Exemplo de uma estrutura de um JSP.....26

Figura 5 - Exemplo da construção de uma solução aleatória.....27

Figura 6 – Segundo exemplo da construção de uma solução aleatória. 27

Figura 7 – Terceiro exemplo da construção de uma solução aleatória. 28

Figura 8 – Exemplo de gerar vizinho troca de operações antes de uma troca.....29

Figura 9 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo um. 29

Figura 10- Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo dois.....30

Figura 11 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo três. 30

Figura 12 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo quatro. 31

Figura 13 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo quatro. 31

Figura 14 - Exemplo de troca entre as máquinas antes da troca passo um.....32

Figura 15 - Exemplo de troca entre as máquinas depois da troca passo dois. 32

Figura 16 - Exemplo de troca entre as máquinas depois da troca passo três.....33

Figura 17 - Exemplo de troca entre as máquinas depois da troca passo quatro.....33

Figura 18- Pseudocódigo da SA.....35

Figura 19 - Representação da ordem de processamento dos Jobs de duas soluções. 36

Figura 20 – Pseudocódigo da CS utilizando SA. 38

Figura 21 – Pseudocódigo da seleção das soluções não dominadas proposta nesse trabalho. 39

Figura 22 – Curva de Pareto da instancia Mk10_B1. 46

LISTA DE TABELAS

Tabela 1 – Combinação da calibração dos valores para os parâmetros. 40

Tabela 2 - Valores definido dos parâmetros. 41

Tabela 3 – Principais resultados obtidos da minimização do makespan na literatura. 41

Tabela 4 - Soluções não dominadas das Instâncias de Brandimarte B1. 42

Tabela 5 - Soluções não dominadas das Instâncias de Brandimarte B1 parte 2. 43

Tabela 6 - Soluções não dominadas das Instâncias de Brandimarte B2. 44

Tabela 7 - Soluções não dominadas das Instâncias de Brandimarte B2. 45

SUMÁRIO

1. INTRODUÇÃO 12

1.1 O PROBLEMA E SUA IMPORTÂNCIA 13

1.2 OBJETIVOS 14

1.2.1 Objetivo geral 14

1.2.2 Objetivos específicos 15

1.3 APRESENTAÇÃO DO TRABALHO 15

2. REVISÃO DE LITERATURA 16

2.1 O PROBLEMA JOB SHOP 16

2.2 O PROBLEMA *JOB SHOP SCHEDULING* FLEXÍVEL 17

***2.3* O PROBLEMA *JOB SHOP* MULTIOBJETIVO 17**

2.4 META-HEURÍSTICA *CLUSTERING SEARCH* 19

3. METODOLOGIA 21

3.1 MODELO MATEMÁTICO PARA O FJSP MULTI-OBJETIVO 21

3.2 INSTÂNCIAS 24

3.3 REPRESENTAÇÃO DE UMA SOLUÇÃO 25

3.3.1 Solução inicial 26

3.4 ESTRUTURA DE VIZINHANÇA 28

3.4.1 N1 - Troca entre operações 28

3.4.2 N2 - Troca de operações entre máquinas 31

3.5 META-HEURÍSTICA CLUSTERING SEARCH 33

3.5.1 Uma meta-heurística geradora de soluções 34

3.5.2 Agrupamento Interativo 35

3.5.3 Analisador de agrupamentos 36

3.5.4 Algoritmo de otimização local 37

3.5.5 Algoritmo CS 37

3.6 SELEÇÃO DAS SOLUÇÕES NÃO-DOMINADAS **38**

4. RESULTADOS 40

5. CONCLUSÕES 47

6. REFERÊNCIAS 48

1. INTRODUÇÃO

No atual ambiente competitivo, para uma indústria acompanhar o crescimento do mercado, torna-se necessário a programação da produção de forma eficiente. As indústrias devem cumprir o prazo estipulado para os clientes, pois clientes insatisfeitos poderão resultar na diminuição da necessidade de produção. Com isso, as empresas devem programar suas atividades de tal forma que os recursos sejam utilizados de maneira eficiente (SINGH; MAHAPATRA, 2016).

A programação das atividades em uma indústria, consiste em determinar o sequenciamento das tarefas (*jobs*) em cada máquina, de forma que minimize algum objetivo. Nesta categoria de problema tem-se um conjunto de tarefas, cada uma formada por uma sequência de operações, onde cada operação é realizada por uma única máquina específica. Para um *job* ser processado ele deverá passar por todas as máquinas com a ordem previamente definida. O objetivo deste problema, denominado *Job Shop Scheduling* (JSP) se resume a designar a ordem das operações de cada j*ob* nas máquinas previamente definidas, visando otimizar algum critério (RODAMMER, 1988).

Considerando a limitação do JSP de possuir uma sequência fixa de operações para cada *job*, surge uma extensão denominada problema *Job Shop Scheduling* Flexível (FJSP). Nesta perspectiva é possível que alguns *jobs* possam ser processados por máquinas alternativas, isto é, um mesmo *job* possa ter variações no sequenciamento da sua produção (HO; TAY, 2004).

O FJSP pode possuir vários critérios diferentes como função objetivo, sendo o tempo de término do processamento da última operação, chamado de *makespan* o mais estudado. Outro objetivo que se torna interessante é o somatório do tempo de atraso na entrega dos produtos (atraso total), entre outros (BISSOLI; AMARAL, 2015; MELO; RONCONI, 2015).

Laguna e Martí (1999) provaram que o JSP pertence à classe NP-Difícil para problemas com mais de duas instâncias de máquinas e conseguintemente, Koulamas (1994) provou por redução que o FJSP também recebe essa classificação. Problemas desta classe já foram solucionados através de métodos exatos e heurísticos. A categoria de métodos exatos consegue garantir que a solução é ótima, porém, tendem

a ser ineficazes em instâncias maiores, não encontrando sequer uma solução que seja viável para o problema (CARVALHO *et al*., 2001). Com base na qualidade dos resultados apresentados na literatura para resolver o FJSP utilizando meta-heurísticas, torna-se interessante a utilização dessa técnica para resolver o problema. (GOMES JÚNIOR, 2007; BISSOLI; AMARAL, 2015; Ho; Tay, 2004).

Entretanto, com a necessidade contínua de melhoria na qualidade no processo de tomada de decisão, juntamente com o objetivo de tornar o modelo mais próximo do real contexto operacional, surge a necessidade de tratar o problema considerando simultaneamente mais de um objetivo. Tais problemas são chamados de problemas multiobjetivo (SINGH; MAHAPATRA, 2016; SCRICH, 1997).

Neste trabalho será abordado o FJSP multiobjetivo, com o objetivo de minimizar o *makespan* e o atraso total das tarefas. Como método de solução é implementada a meta-heurística *Clustering Search* (CS), considerando que a mesma vem apresentando bons resultados para outros problemas de otimização combinatória, e que ainda foi encontrada nenhuma aplicação da CS ao FJSP (RIBEIRO FILHO, NAGANO LORENA, 2006; ARAÚJO, MAURI, 2013; OLIVEIRA; CHAVES, LORENA, 2013).

1.1 O PROBLEMA E SUA IMPORTÂNCIA

A medida em que a competitividade das empresas no mercado global vem se intensificando, torna-se necessário o uso de estratégias que possibilitam que uma empresa predomine às outras, possibilitando assim, sua sobrevivência no mercado (SCRICH, 1997).

Entre as estratégias que possibilitam uma indústria expandir seu mercado, uma alternativa pode ser a adição de novos produtos, que podem demandar flexibilidade na produção. Essa flexibilidade se justifica, em sistemas de linha de produção, em aproveitar ao máximo a infraestrutura existente sem a necessidade de grandes investimentos para produzir novos produtos em paralelo (CHIAVENATO, 2004; SCRICH, 1997).

Para uma empresa se manter no mercado, ela deve possuir uma margem razoável de lucro e paralelamente a isso, aumentar o preço de seu produto poder tornar-se

inviável, devido à concorrência. Com isso, torna-se interessante utilizar-se da estratégia de redução do custo do produto (CHIAVENATO, 2004).

Fatores como o tempo de produção ou mesmo atrasos são importantes em ambientes produtivos, pois implicam diretamente no custo final do produto. Ao melhorar esses fatores, há uma tendência de se obter uma produção mais enxuta, consequentemente, mais barata. Proporcionando uma melhoria na satisfação dos clientes, pois um produto entregue sem atraso, tende a fidelizá-los. Com isso, torna-se fundamental o eficiente sequenciamento de produção para a competitividade das empresas (CHIAVENATO, 2004).

É importante ressaltar que tais fatores relatados podem ser conflitantes, isto é, ao priorizar um, pode-se afetar o outro. Assim, torna-se interessante uma análise que considere ambos fatores. Métodos que se encaixam nessa abordagem são chamados de multiobjetivo (CARVALHO, 2015).

Dessa forma, o estudo de métodos eficientes para resolução do problema *Job Shop Scheduling* multiobjetivo flexível se torna importante, para auxiliar a tomada de decisão em ambientes de sequenciamento de produção, conforme descrito anteriormente.

1.2 OBJETIVOS

A seguir são descritos os objetivos gerais e específicos relacionados a este trabalho.

1.2.1 Objetivo geral

O objetivo geral deste projeto é desenvolver a adaptação da meta-heurística *Clustering Search* (CS) para gerar soluções para o *Job Shop Flexível* (FJSP) Multiobjetivo, e assim obter a curva de Pareto.

1.2.2 Objetivos específicos

Com o intuito de atingir o objetivo principal, alguns objetivos específicos são requeridos, entre eles:

a) Revisar o FJSP com versões multiobjetivo;
b) Adaptar o modelo matemático, para representar o FJSP Bi-Objetivo;
c) Definir o conjunto de instâncias para realizar de experimentos computacionais;
d) Implementar a meta-heurística *Clustering Search* utilizando o *Simulated Annealing* como geradora de solução;
e) Gerar a curva de Pareto;
f) Discutir os experimentos computacionais;

1.3 APRESENTAÇÃO DO TRABALHO

O trabalho está dividido da seguinte forma: no Capítulo 2 é apresentada uma revisão bibliográfica dos principais e mais recentes trabalhos da literatura. O Capítulo 3 detalha como que a CS será aplicado no FJSP. Os resultados são descritos no Capítulo 4. O Capítulo 5 apresenta a conclusão, no Capítulo 6, são enumeradas as referências utilizadas nesse trabalho.

2. REVISÃO DE LITERATURA

2.1 O PROBLEMA JOB SHOP

Bissoli e Amaral (2015) descrevem o problema *Job Shop Scheduling* (JSP) da seguinte maneira: são dados um conjunto de m máquinas, e um conjunto de n *jobs*, sendo que cada *job* é composto por uma sequência de operações que devem ser executadas em uma dada ordem, isto é, respeitando uma regra de precessão. Cada uma das operações é atribuída a uma máquina específica com um tempo fixo de processamento e cada máquina só pode processar uma operação por vez, não podendo interromper uma operação em execução.

De acordo com Rodammer (1988), o problema clássico resume-se em determinar o sequenciamento dos *jobs* em cada máquina, de forma que minimize o tempo de término da última operação completada, sendo este objetivo conhecido como *makespan*.

Arenales *et al.* (2007) complementa que além do objetivo *makespan*, existem também outros objetivos como a minimização do tempo do fluxo total, tais como ao somatório dos tempos de término das tarefas, minimização do atraso máximo, minimizar a tarefa que tem o maior tempo de atraso e minimização do atraso total que é o somatório dos atrasos das tarefas.

Laguna e Martí (1999) demonstram que se o objetivo do JSP for a minimização do *makespan,* ele será considerado um problema NP-Difícil. Desta forma, não se conhece um algoritmo que os resolva em um tempo polinomial. Por isso, é compreensível sacrificar a certeza da otimalidade, para ter como recompensa uma solução de boa qualidade (CARVALHO *et al.*, 2001).

Alguns estudos abordam variações do JSP, de modo a se aproximar de um problema real específico. Por exemplo, o problema denominado *Flow Shop* considera que as n tarefas tem a mesma sequência de máquinas. Outra abordagem, o *Job shop* com interrupção, existe situações onde uma tarefa com maior prioridade torna-se disponível para o processamento, esse modelo permite que uma máquina interrompa um processo antes de finaliza-lo (ARENALES *et al.,* 2007). A seguir, é descrito o *Job Shop Scheduling* Flexível, que é a variação explorada nesse trabalho.

2.2 O PROBLEMA *JOB SHOP SCHEDULING* FLEXÍVEL

De acordo com Ho e Tay (2004), o JSP é limitado por atribuir uma única máquina a um *job*, assim, surgiu o *Job Shop Scheduling* Flexível (FJSP), que, além de estender o JSP, permite que cada operação pode ser processada por mais de uma máquina.

Com isso, Carvalho (2015) complementa que o FJSP permite a disponibilidade de mais de um recurso (máquina) para realizar uma tarefa. Estando assim, mais próximo de modelos de aplicações do mundo real.

Existe um grande número de trabalhos na literatura que propõem métodos para solucionar o FJSP, conforme pode ser observado na revisão realizada por Chaudhry e Khan (2016). Em relação aos melhores resultados conhecidos, Gonzáles *et al*. (2015) propôs uma versão híbrida da meta-heurística *Scatter Search* que é considerado o estado da arte para o problema.

Ainda com o objetivo de aproximar o problema FJSP ao real contexto operacional, alguns trabalhos realizaram abordagens que visam adicionar novos objetivos ao problema clássico, de modo a proporcionar uma melhoria na qualidade no processo de tomada de decisão (MOREIRA *et al*., 2008; CARVALHO, 2015).

2.3 O PROBLEMA *JOB SHOP* MULTIOBJETIVO

Em problemas de otimização multiobjetivo, é esperado a satisfação de todos os objetivos, contudo, achar essa solução não é trivial, pois os objetivos tendem a conflitar entre si, isto é, ao melhorar um objetivo, pode ocasionar a piora do outro (JOZEFOWIEZ *et al*., 2008).

Carvalho (2015) classificou tipos de abordagens de problemas multiobjetivo:

- Objetivo único: é selecionado um objetivo e os demais são aplicados na restrição do problema;
- Método de função utilitária: a função objetivo é linearizada, sendo que cada objetivo recebe um valor multiplicador, que é classificado de acordo com a importância do objetivo. O foco dessa abordagem é utilizar o conhecimento do nível estratégico de modo a ponderar a função objetivo conforme a necessidade;

- Ótimo de Pareto: Considera que geralmente os objetivos de um problema multiobjetivo são conflitantes, ou seja, ao melhorar um objetivo, pode-se afetar e/ou piorar o outro. Com isso, não existe uma única solução que seja ótima para todos os objetivos. Nessa situação, busca-se um conjunto de soluções com uma qualidade semelhante que satisfaça a restrição do problema. Essa qualidade semelhante seriam as soluções não dominadas.

Para entender melhor as classificações de tipos de abordagens de problemas multiobjetivo, Jozefowiez (2008) exemplifica o que é a relação de dominância de funções:

- Dominância: sejam x e y, duas soluções diferentes, que pertencem ao conjunto S, $\{f_1(x), f_2(x) \ldots, f_n(x)\}$ de funções objetivo, onde n é o número de objetivos. Logo, x domina y em um problema de minimização, se e somente se, $f_1(x) \leq f_1(y), f_2(x) \leq f_2(y) \ldots f_n(x) \leq f_n(y)$ e existe pelo menos um objetivo c, tal que $1 \leq c \leq n$, para um $f_c(x) < f_c(y)$.
- Não dominados: dado um conjunto solução S, as soluções não dominadas é o subconjunto de S, que contém soluções que não são dominadas por nenhuma outra solução.

Essas classes de problemas podem ser solucionadas através de métodos exatos, a partir de modelos de programação matemática. Entretanto, tais métodos tendem a ser ineficazes em instâncias que possuem uma dimensão maior, isto é, não conseguem obter a solução ótima, ou mesmo uma solução viável em tempo computacional hábil, o que é comum se considerarmos aplicações reais do FJSP (GOMES JÚNIOR, 2007). Dessa forma, surge a motivação de utilizar métodos heurísticos, que geram soluções por aproximação para a resolução desses problemas.

Na literatura são encontradas diferentes abordagens multiobjetivas para o FJSP. Yuan e Xu (2015) fizeram uma extensa revisão de literatura referente a FJSP multiobjetivo, que trata dos objetivos *makespan*, e critérios relacionados a carga de trabalho.

Moreira *et al*. (2008) propuseram três versões de algoritmos genéticos para minimizar os objetivos *makespan* e o atraso total das tarefas. Os resultados foram comparados com a literatura, obtendo assim, um maior número de soluções não dominadas em 80% dos problemas com o algoritmo genético com *path relinking*.

Rabiee *et al.* (2012) implementaram a meta-heurística *nondominated sorting genetic algorithm* (NSGA-II), *non-dominated ranked genetic algorithm* (NRGA), *multiobjective genetic algorithm* (MOGA) e Pareto *archive evolutionary strategy* (PAES), para o FJSP multiobjetivo minimizando o *makespan*. As soluções obtidas pelo PAES foram melhores que as dos outros algoritmos.

Carvalho (2015) considera *makespan*, carga de trabalho total e carga de trabalho máxima como objetivos tradicionais, e incluiu novos objetivos ao FJSP multiobjetivo, sendo eles a ociosidade total e ociosidade efetiva total.

Mais recentemente, Carvalho (2015) apresentou um algoritmo *Particle Swarm Optimization* (PSO) com diversidade para minimizar o *makespan*, carga de trabalho total, representado por o somatório do tempo de trabalho de todas as máquinas e a carga de trabalho máxima, onde na carga de trabalho o tempo ocioso é desconsiderado. Seu método proporcionou novos melhores resultados nas instâncias Mk01 e Mk02, proposta por Brandimarte (1993).

Considerando que a proposta apresentada por Moreira *et al.* (2008) ainda não possui aplicação ao FJSP, juntamente com a aplicação da meta-heurística *Clustering Search* (CS), torna-se interessante o estudo dessa abordagem. Dessa forma, a seguir apresenta-se a meta-heurística *Clustering Search*, implementada neste trabalho.

2.4 META-HEURÍSTICA *CLUSTERING SEARCH*

Empregar estratégias que utilizam heurísticas para a resolução de problemas de otimização combinatória se tornam interessante, pois proporcionam produzir soluções com boa qualidade ou até obter a solução ótima, em um tempo computacional razoável (GOMES JÚNIOR, 2007).

Até o presente momento não foram encontradas abordagens realizadas que aplique a meta-heurística *Clustering Search* (CS) ao FJSP multiobjetivo. Dessa forma, torna-se interessante o estudo de sua aplicação para o FJSP multiobjetivo, considerando que a CS apresentou resultados satisfatórios quando aplicado em vários problemas de otimização combinatória conforme explanado a seguir.

Araújo e Mauri (2013) propuseram uma meta-heurística *Clustering Search* para resolver o problema de rotação de culturas com restrições de adjacências (PRC-A). O

algoritmo apresentou-se como uma nova e boa alternativa para resolução do PRC-A considerando o relato de uma melhora em até 65,36% em relação ao algoritmo genético proposto por Aliano Filho (2012).

Rabello *et al*. (2013) utilizando uma meta-heurística híbrida CS para o Problema de Rotulação Cartográfica de Pontos (PRCP), obtiveram resultados superiores para instância de 1000 e 13206 pontos aos conhecidos na literatura, deixando visível a maior quantidade de números de rótulos sem sobreposições.

Ribeiro Filho *et al.* (2006) propuseram um algoritmo *Evolutionary Clustering Search* (ECS) para solucionar o *flow shop*, comparado com a literatura, essa meta-heurística encontrou a melhor solução em 91,1% dos problemas no total de 82, sendo que 59 dessas soluções ainda não tinham sido encontradas.

Oliveira *et al*. (2013) elaboraram três variações do algoritmo CS, utilizando as meta-heurísticas como geradora de soluções, Algoritmo Genético (AG), *Simulated Annealing* (SA) e *Iterated Local Search* (ILS). Utilizando a meta-heurística ILS, o método conseguiu encontrar 10 das melhores soluções conhecidas dos 25 problemas da literatura, utilizando o AG, o algoritmo conseguiu se igualar a 13 da literatura e utilizando o SA a CS foi capaz de encontrar 14 soluções equivalente as 25 conhecidas. Logo torna-se motivacional o uso do SA, como gerador de solução para a CS.

O próximo capitulo explicará como a CS foi aplicado ao JSP bi-objetivo.

3. METODOLOGIA

Para resolver o FJSP computacionalmente, primeiramente é preciso abstrair a realidade do problema para um conjunto de equações que o representam de forma qualitativa. Para isso, é preciso definir quais hipóteses serão utilizadas no modelo para representar o sistema real (MAURI, 2005).

Esse processo de transformar um problema da vida real em um modelo matemático é definido como Programação Linear (PL) (SIERKSMA, 1996). Esse modelo matemático é representado por uma ou mais funções objetivo, que podem ser de minimização ou maximização, seguidos por um conjunto finito de restrições. Sendo essas funções Lineares, e estritamente inteiras, a programação linear é chamada de Programação Linear Inteira (PLI), ou simplesmente Programação Inteira. Dentro da literatura foram encontrados modelos de Programação Linear Inteira Mista (PLIM), para modelar o FJSP.

3.1 MODELO MATEMÁTICO PARA O FJSP MULTI-OBJETIVO

Com o intuito de buscar soluções ótimas, Melo e Ronconi (2015) propõem o Modelo de Programação Linear Inteira Mista (PLIM) para o FJSP.

Sejam os índices J_i, um *job* que pertence ao conjunto de *jobs* J, M um conjunto de máquinas, M_k uma máquina que pertence ao conjunto M, O o conjunto de operação de um *job*, O_i a representação do conjunto de operação do *job* i, O_{ij} a operação j pertencente ao *job* i, M_j o conjunto de máquinas alternativas da operação do *job*, $M_j \cap M_{J'}$ o conjunto de máquinas onde ambas as operações j e j' podem ser processadas e P_{ij} o conjunto de precedência de operações de um *job*.

Os parâmetros: t_{ijk} correspondente ao tempo de processamento da operação O_{ij} na máquina k, d_i sendo a data de entrega do *job* i e L a operação com o maior tempo de processamento.

Para entender melhor o modelo é importante entender as variáveis de decisão, sendo elas: x_{ijk} um valor binário que corresponde se a máquina k foi selecionada para atender a operação O_{ij}, S_{ijk} inicio de processamento da operação O_{ij} na máquina k, C_{ijk} instante de término do processamento da operação O_{ij} na máquina k, $Y_{iji'j'k}$

é a representação binária que descreve se a O_{ij} precede a $O_{i'j'}$, C_i representa o instante de término de processamento do *job* i e o T_i o atraso do *job* i.

Por fim, é descrito o PLIM do FJSP com o objetivo de minimizar o atraso total das tarefas proposto por Melo e Ronconi, (2015):

Minimizar

$$\sum_{i \in J} T_i \tag{1}$$

Sujeito a:

$$\sum_{k \in M_j} X_{ijk} = 1 \qquad \forall\, i \in j, \forall\, j \in O_i \tag{2}$$

$$S_{ijk} \leq X_{ijk} * L \qquad \forall\, i \in j, \forall\, j \in O_i, \forall\, k \in M_j \tag{3}$$

$$C_{ijk} \leq X_{ijk} * L \qquad \forall\, i \in j, \forall\, j \in O_i, \forall\, k \in M_j \tag{4}$$

$$C_{ijk} \geq S_{ijk} + t_{ijk} - (1 - X_{ijk}) * L \qquad \forall\, i \in j, \forall\, j \in O_i, \forall\, k \in M_j \tag{5}$$

$$S_{ijk} \geq C_{i'j'k} - (Y_{iji'j'k}) * L \qquad \forall\, i < i', \forall\, j \in O_i, \forall\, j' \in O_{i'}, \forall\, k \in M_j \cap M_{j'} \tag{6}$$

$$S_{i'j'k} \geq C_{ijk} - (1 - Y_{iji'j'k}) * L \qquad \forall\, i < i', \forall\, j \in O_i, \forall\, j' \in O_{i'}, \forall\, k \in M_j \cap M_{j'} \tag{7}$$

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{k \in M_j} C_{i'j'k} \qquad \forall\, i \in j, \forall\, j \in O_i, \forall\, O_{ij} \in P_{ij} \tag{8}$$

$$C_i \geq \sum_{k \in M_j} C_{ijk} \qquad \forall\, i \in J, \forall\, J \in O_i \tag{9}$$

$$T_i \geq C_i - d_i \qquad \forall\, i \in J \tag{10}$$

$$X_{ijk} \in \in \{0,1\} \qquad \forall\, i \in j, \forall\, j \in O_i, \forall\, k \in M_j \tag{11}$$

$$S_{ijk} \geq 0 \qquad \forall\, i \in j, \forall\, j \in O_i, \forall\, k \in M_j \tag{12}$$

$C_{ijk} \geq 0$ $\forall\, i \,\epsilon\, j, \forall\, j \,\epsilon\, O_i, \forall\, k \,\epsilon\, M_j$ (13)

$Y_{iji'j'k} \in \{0,1\}$ $\forall\, i < i'; \forall\, j \in O_i;\ \forall\, j' \in O_{i'}, \forall\, k \in M_j \cap M_{j'}$ (14)

$C_i \geq 0$ $\forall\, i \,\epsilon\, j$ (15)

$T_i \geq 0$ $\forall\, i \,\epsilon\, j$ (16)

A função objetivo (1) visa minimizar o atraso total. As restrições do tipo (2) garantem que cada operação só terá uma única máquina para processa-la, enquanto as restrições (3) e (4) certificam que, se uma operação não estiver alocada a uma máquina, os instantes de início e término de processamento nessa máquina sejam zero. As restrições (5) asseguram que o instante de término de processamento de uma operação atribuída a uma máquina, será maior ou igual a soma do que seu instante de início com o tempo de processamento desta máquina. As restrições (6 e 7) impedem que uma máquina processe mais de uma operação simultaneamente. As restrições (8) garantem a precedência das operações, enquanto as restrições (9) certificam que o tempo de término de processamento do *job* seja maior ou igual que o tempo de término das operações. As restrições (10) certificam que o *job* não será entregue antes de ter todas as operações finalizadas enquanto as restrições (11 a 16) definem o domínio das variáveis. Por fim as restrições (11 a 14) tratam a integridade das variáveis.

Para adaptar o modelo proposto por Melo e Ronconi, (2015), para o modelo bi-objetivo é adicionado uma nova função objetivo (17), que minimiza o *makespan*.

Minimizar

$\mathrm{Max}(C_i)$ (17)

Com base no modelo matemático apresentado, é destacado a complexidade do problema, pois não é interessante entregar somente uma solução, e sim um conjunto pois se trabalha com mais de um objetivo. Com isso, conseguimos dar um poder maior de decisão para o interessado no problema, aproximando o modelo mais ainda a

realidade, possibilitando uma visão mais estratégica. Na seção seguinte, é abordada a meta-heurística selecionada, para solucionar esse modelo matemático.

3.2 INSTÂNCIAS

As instâncias utilizadas nesse trabalho foram todas as propostas por Brandimarte (1993). Por não considerar data de entrega, as mesmas foram adaptadas de forma similar às datas geradas por Moreira *et al.* (2008), conforme apresentado nas equações 18,19 e 20.

$$d_j \qquad Piso\,(\,\beta\ *\sum_i \bar{t}_{ij}\,) \tag{18}$$

$$\beta_1 = ((J * M)/1000) + 0{,}5 \tag{19}$$

β

$$\beta_2 = \beta_1 + 1 \tag{20}$$

De acordo com as equações 18,19 e 20, os parâmetros β_1 e β_2 são utilizados para determinar a data de entrega, sendo β_2 uma forma mais folgada de gerar as datas de entrega. A data de entrega do *job* j é referenciado em dj, sendo $\bar{t}_{ij}$ a média do tempo de processamento das operações do *job* j. Para evitar problemas de arredondamento ou diferença ao considerar casas decimais, nesse trabalho é proposto truncar os valores da data de entrega.

Para cada instância de Brandimarte (1993) são geradas duas novas instâncias, uma com flexibilidade β_1 e outra β_2, conforme Figura 1.

```
10  6  2
6 13 2 1 5 3 4 3 5 3 3 5 2 1 2 3 4 6 2 3 6 5 2 6 1 1 1 3 1 3 6 6 3 6 4 3
5 10 1 2 6 1 3 1 1 1 2 2 2 6 4 6 3 6 5 2 6 1 1
5 14 1 2 6 2 3 4 6 2 3 6 5 2 6 1 1 3 3 4 2 6 6 6 2 1 1 5 5
5 10 3 6 5 2 6 1 1 1 2 6 1 3 1 3 5 3 3 5 2 1 2 3 4 6 2
6 16 3 5 3 3 5 2 1 3 6 5 2 6 1 1 1 2 6 2 1 5 3 4 2 2 6 4 6 3 3 4 2 6 6 6
6 12 2 3 4 6 2 1 1 2 3 3 4 2 6 6 6 1 2 6 3 6 5 2 6 1 1 2 1 3 4 2
5 11 1 6 1 2 1 3 4 2 3 3 4 2 6 6 6 3 2 6 5 1 1 6 1 3 1
5 16 2 3 4 6 2 3 3 4 2 6 6 6 3 6 5 2 6 1 1 1 2 6 2 2 6 4 6
6 13 1 6 1 2 1 1 5 5 3 6 6 3 6 4 3 1 1 2 3 3 4 2 6 6 6 2 2 6 4 6
6 12 2 3 4 6 2 3 3 4 2 6 6 6 3 5 3 3 5 2 1 1 6 1 2 2 6 4 6 2 1 3 4 2
```

Figura 1 – Nova estrutura da instância.

Na primeira linha tem-se a quantidade de *jobs* seguido pela quantidade de máquinas e o grau de flexibilidade do problema. Na segunda linha em diante temos uma matriz onde cada linha representa um *job*. Na primeira coluna é informada a quantidade de operações de cada *job*, representado em verde. Na segunda coluna obtém-se a data de entrega, em azul. As próximas colunas são as informações de cada operação do *job*, de forma sequencial, onde a primeira informação significa quantas máquinas podem processar aquela operação, em roxo, seguido pelo número da máquina e o tempo para essa máquina processar esta operação. Tal formato é destacado na Figura 2.

6	13	2	1	5	3	4	3	5	3	3	5	2	1	2	3	4	6	2	3	6	5	2	6	1	1	1	3	1	3	6	6	3	6	4	3
5	10	1	2	6	1	3	1	1	1	2	2	2	6	4	6	3	6	5	2	6	1	1													
5	14	1	2	6	2	3	4	6	2	3	6	5	2	6	1	1	3	3	4	2	6	6	6	2	1	1	5	5							
5	10	3	6	5	2	6	1	1	1	2	6	1	3	1	3	5	3	3	5	2	1	2	3	4	6	2									
6	16	3	5	3	3	5	2	1	3	6	5	2	6	1	1	1	2	6	2	1	5	3	4	2	2	6	4	6	3	3	4	2	6	6	6
6	12	2	3	4	6	2	1	1	2	3	3	4	2	6	6	6	1	2	6	3	6	5	2	6	1	1	2	1	3	4	2				
5	11	1	6	1	2	1	3	4	2	3	3	4	2	6	6	6	3	2	6	5	1	1	6	1	3	1									
5	16	2	3	4	6	2	3	3	4	2	6	6	6	3	6	5	2	6	1	1	1	2	6	2	2	6	4	6							
6	13	1	6	1	2	1	1	5	5	3	6	6	3	6	4	3	1	1	2	3	3	4	2	6	6	6	2	2	6	4	6				
6	12	2	3	4	6	2	3	3	4	2	6	6	6	3	5	3	3	5	2	1	1	6	1	2	2	6	4	6	2	1	3	4	2		

Figura 2 – Representação colorida de parte da instância Mk01.

3.3 REPRESENTAÇÃO DE UMA SOLUÇÃO

A solução é composta por duas matrizes $k \times \max(jobs)$ e $i \times \max(máquinas)$ onde i representa a quantidade de *jobs* e k a quantidade de máquinas, essas matrizes são preenchidas pelo tempo de término do processamento dos *jobs* na máquina respectiva. Uma matriz de $2 \times y$, onde y é o somatório das operações de todos os *jobs*. Na linha 1 de y guarda-se a ordem que os *jobs* foram inseridos, e na outra linha é preenchida com as máquinas sorteadas para processar esses *jobs*, como ilustrado na Figura 3.

	Posição		
Máquinas	0	1	2
M1	0		
M2	0		

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0		
J2	0		

Jobs					
Máquinas					

Figura 3 – Exemplo da representação de uma solução

3.3.1 Solução inicial

Com o objetivo de fornecer uma solução inicial a CS, inicialmente é criado uma solução para o problema. Esta construção é realizada de forma aleatória, tendo como exemplo a Figura 4.

Jobs	Operações,Tempo	
1	M1,6	M2,10
2	M2,6	

Figura 4 - Exemplo de uma estrutura de um JSP

A Figura 4 representa um problema com dois *jobs* onde o *job* j_1 é composto por duas operações, a primeira é processada pela máquina m_1 e seu tempo de processamento é 6. A segunda é processada pela máquina m_2 e seu tempo de processamento é 10. Já o *job* j_2 é composto apenas por uma operação processada pela máquina m_2 e seu tempo de processamento é 6.

A construção da solução inicial é realizada de forma parcialmente aleatória, inicialmente é criado um vetor para auxiliar o sorteio, para cada operação, é inserido o número do *job* respectivo no vetor, neste exemplo, o vetor terá o seu valor preenchido como $v = [1,1,2]$, então é sorteado um elemento do vetor, a construtiva inicia-se sorteando um *job* aleatório, por exemplo, j_1 . Então, é escolhida sua operação respeitando a ordem de precedência. Com isso, inicialmente é escolhida a operação O_{11}. É escolhida então, a máquina com menor *makespan* dentre as quais podem

processar esta operação, neste caso a m_1. Logo, seleciona-se o maior tempo entre o *job* selecionado e essa máquina (em vermelho na Figura 5). Esse tempo é acrescido pelo tempo de processamento deste *job* nesta máquina, e é inserido na matriz de *jobs*, máquinas e nos vetores de ordem de inserção.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0		

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	Valor inserido na interação

	Posição		
Jobs	0	1	2
J1	0	6	
J2	0		

Jobs	1				
Máquinas	1				

Figura 5 - Exemplo da construção de uma solução aleatória.

Continuando o exemplo, foi sorteado o *job* j_1. Sabendo que a operação O_{11} já foi processada, logo, será processada a operação O_{12}. É selecionado o maior tempo entre o *job* sorteado e a máquina que o processará, no caso 6. Adicionado com o tempo de processamento da operação O_{12} foi obtido 16, esse valor é inserido na solução apresentada na Figura 6.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	valor inserido na interação

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0		

Jobs	1	1			
Máquinas	1	2			

Figura 6 – Segundo exemplo da construção de uma solução aleatória.

Por fim, é selecionado o *job* j_2. O maior tempo entre o *job* selecionado e a máquina que o processará é 16, acrescidos ao tempo de processamento foi obtido 22, conforme demonstrado na Figura 7.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	22

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	valor inserido na interação

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0	22	

Jobs	1	1	2		
Máquinas	1	2	2		

Figura 7 – Terceiro exemplo da construção de uma solução aleatória.

3.4 ESTRUTURA DE VIZINHANÇA

De modo a buscar melhores soluções, a CS possui a característica de explorar a vizinhança de forma iterativa, realizando trocas entre os vizinhos. Duas formas de explorar a vizinhança foram utilizadas: N1 e N2 que são descritas a seguir.

3.4.1 N1 - Troca entre operações

A estratégia dessa vizinhança constitui-se em alterar a ordem de processamento das operações na máquina. Primeiramente, é selecionada aleatoriamente uma máquina, em seguida é sorteada uma das operações processadas por essa máquina excluindo a primeira. Essa operação é trocada com a operação anterior a ela Figura 8.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	22

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0	22	

Jobs	1	1	2		
Máquinas	1	2	2		

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	valor inserido na interação

Figura 8 – Exemplo de gerar vizinho troca de operações antes de uma troca

A Figura 8 representa execução da vizinhança N1, foi sorteada a operação $m_{2,2}$ da máquina m_2, o próximo passo é desfazer a solução até que a quantidade de operações processadas pela máquina sorteada seja inferior ao número da operação sorteada Figura 9 e Figura 14.

Antes da remoção:

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	22 ✖

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0	22 ✖	

Jobs	1	1	2 ✖		
Máquinas	1	2	2 ✖		

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	valor inserido na interação

Figura 9 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo um.

Depois da remoção:

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	valor inserido na interação

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0		

Jobs	1	1	2		
Máquinas	1	2	2		

Figura 10- Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo dois.

Logo após, a solução continua sendo desfeita até que a quantidade de operações da máquina no caso a m_2 decremente em um, porém a cada operação removida, o job selecionado para ser para ser aforado é priorizado, isto é, ele é deslocado para a direita Figura 11.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	

Cinza	index da matriz
Amarelo	valor definido na inicialização
Azul	valor inserido na interação

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0		

Jobs	1	1	2		
Máquinas	1	2	2		

Figura 11 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo três.

Posteriormente a solução é reconstruída utilizando os vetores que guardam a ordem que o *jobs* e maquinas foram inseridos Figura 12 e Figura 13.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	✚ 6	

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0	6	
J2	0	✚ 6	

Jobs	1	✚ 2	1		
Máquinas	1	✚ 2	2		

Figura 12 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo quatro.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	6	✚ 16

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0	6	✚ 16
J2	0	6	

Jobs	1	2	✚ 1		
Máquinas	1	2	✚ 2		

Figura 13 - Exemplo de gerar vizinho troca de operações depois da troca, destruição da solução passo quatro.

3.4.2 N2 - Troca de operações entre máquinas

Esse método busca uma operação que tenha grau de flexibilidade maior que um, ou seja, que possa ser processada por mais de uma máquina. Em um vetor pré-processado, isto é, contendo todas as operações desse perfil, é sorteada uma operação. Esta operação é identificada na solução, e é alterada a máquina que a processa.

Seguindo o exemplo da Figura 7, se a operação O_{21} suportasse ser processada por mais de uma máquina e a m_1 fosse uma máquina alternativa para processa-la, esse método trocaria essa operação de máquina, exibido na Figura 14.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	22

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0	22	

Jobs	1	1	2		
Máquinas	1	2	2		

Figura 14 - Exemplo de troca entre as máquinas antes da troca passo um.

Para isso a solução será desfeita até que fosse retirada essa operação da matriz de *Jobs*, neste caso, a operação um do job 2 (O_{21}) Figura 15.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	22 ✖

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0	22 ✖	

Jobs	1	1	2 ✖		
Máquinas	1	2	2 ✖		

Figura 15 - Exemplo de troca entre as máquinas depois da troca passo dois.

Após a operação O_{21} ser removida, é realizado a troca da máquina que a processa, e assim a solução é reconstruída novamente Figura 16 e Figura 17.

	Posição		
Máquinas	0	1	2
M1	0	6	
M2	0	16	

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0		

Jobs	1	1	2		
Máquinas	1	2	1		

Figura 16 - Exemplo de troca entre as máquinas depois da troca passo três.

	Posição		
Máquinas	0	1	2
M1	0	6	12
M2	0	16	

Cinza	index da matriz
Amarelo	valor definido na inicialização

	Posição		
Jobs	0	1	2
J1	0	6	16
J2	0	12	

Jobs	1	1	2		
Máquinas	1	2	1		

Figura 17 - Exemplo de troca entre as máquinas depois da troca passo quatro.

3.5 META-HEURÍSTICA CLUSTERING SEARCH

A CS é uma proposta genérica do *Evolutionary Clustering Search* (ECS). A principal diferença entre eles é que a ECS utiliza um algoritmo evolucionário para gerar soluções, enquanto a CS pode utilizar outras meta-heurísticas, evolucionárias ou não, dentre elas *Simulated Annealing* (SA), ILS, GRASP entre outras (OLIVEIRA *et al.*, 2013).

A CS é um algoritmo híbrido interativo que procura dividir o espaço de busca para localizar regiões promissoras, realizando assim, uma pesquisa mais profunda nessas regiões. (OLIVEIRA *et al.*, 2013).

Cada espaço de busca é conhecido como *cluster*, e este é definido por um conjunto de atributos $C = \{c; v; r\}$ onde: c representa o centro do *cluster*, v o volume do *cluster* e r o índice de ineficácia. O centro do *cluster* é representado pela melhor solução presente em C, o volume indica a quantidade de soluções agrupadas em C, e o índice de ineficácia indica a quantidade de interações realizadas no centro do *cluster* onde a busca local não apresentou melhora na solução.

Segundo Oliveira *et al*. (2013), a CS pode ser entendido em quatro partes independentes. Na seção seguinte aborda-se cada um desses componentes.

3.5.1 Uma meta-heurística geradora de soluções

Inicialmente os *clusters* são preenchidos por um algoritmo construtivo aleatório, e a cada interação da CS, a meta-heurística fornece uma solução para se adequar a um *cluster*. Com base no desempenho do SA comparado com outras meta-heurísticas, este trabalho utiliza a meta-heurística *Simulated Annealing* (SA) como geradora de soluções para a CS.

A SA foi proposta por Kirkpatrick *et al.* (1983), e se baseia na ideia de *Annealing*, processo utilizado para fundir metais, onde o metal é aquecido a uma temperatura elevada e em seguida é resfriado lentamente, tornando o produto cada vez menos flexível e mais homogêneo.

O seu processo de busca pode ser dividido em duas estratégias, a primeira é a estratégia de caminhos aleatórios, onde o objetivo é explorar o espaço de busca, análogo ao *Annealing* no momento onde a temperatura está alta. Entretanto, quando a temperatura diminui, esse comportamento é decrescido, diminuindo a probabilidade de aceitar uma solução pior. Nesta etapa estamos interessados em definir o metal, convergindo a solução a um ótimo local, esta fase é conhecida como melhoramento interativo (HAESER, GOMES–RUGGIERO, 2006)

Em problemas de minimização, o SA recebe por parâmetro uma solução inicial s. essa solução é guardada em s^*, variável responsável por guardar a melhor solução. Em

seguida, gera-se aleatoriamente um vizinho de s, representado por s' e posteriormente é calculado Δ, que é a diferença de s e s'. Se Δ for menor que zero, ou seja, se a solução melhorou, ela é guardada em s e é verificado se s' é melhor que s^*(melhor solução obtida até o momento). Caso afirmativo, s* é atualizada com o valor de s'. Porém, se Δ não for menor que zero, um número real aleatório é definido no intervalo fechado de zero a um e se esse valor for maior que $e^{(-\Delta/T)}$, onde T representa a temperatura, que inicialmente é atribuída por um valor alto, s irá aceitar uma solução pior. Todas essas ocorrências são repetidas até que um número de interações pré-definidas seja atingido e assim, a temperatura é decrementada. Esses processos são repetidos até que a temperatura seja menor ou igual que a temperatura de congelamento, que é pré-definida no algoritmo. A Figura *18* apresenta o pseudocódigo da SA.

```
1.  Dado (α, SA_max, T_0, T_c e s)
2.  s* ← s   iteracao ← 0
3.  tempCorrente ← T_0            {Temperatura corrente}
4.  ENQUANTO tempCorrente > T_C FAÇA
5.     ENQUANTO iteracao < SA_max FAÇA
6.     iteracao ← iteracao + 1
7.     s' ← N(s)   Δ ← f(s') - f(s)
8.        SE Δ < 0 ENTÃO
9.          s ← s'
10.         SE (f(s') < f(s*)) ENTÃO
11.            s* ← s'
12.         FIM-SE;
13.       SENÃO
14.         considere x ∈ [0,1]
15.         SE (x < e^(-Δ/T)) ENTÃO
16.            s ← s'
17.         FIM-SE
18.       FIM-SE
19.    FIM-ENQUANTO
20.    tempCorrente ← α * tempCorrente  iteracao ← 0
21. FIM-ENQUANTO
22. Retornar s*
```

Figura 18- Pseudocódigo da SA.

Adaptado de Mauri (2005).

3.5.2 Agrupamento Interativo

O objetivo dessa etapa é identificar qual *cluster* a solução é mais similar, essa similaridade é obtida através do cálculo da distância de *Hamming (*HAMMING, 1950), onde é contabilizado o número de posições que as soluções se diferenciam entre si. Ao final da comparação, a solução é atribuída ao *cluster* que esteja menos distante, com isso, o volume do *cluster* v_i é incrementado, logo após, é comparado a solução

do centro do *cluster* com a solução gerada pelo SA, se a solução gerada pelo SA for melhor ela vira o centro do *cluster*.

A Figura 19 exibe a programação em duas soluções distintas, elas se diferenciam na máquina que atende o último *job*, a distância de *Hamming* entre as duas soluções é calculada como o valor um.

solução s_1

Jobs		1	1	2	2
Máquinas		1	2	1	2

solução s_2

Jobs		1	1	2	2
Máquinas		1	2	1	1

$$d_{\text{Ham}}(s_1, s_2) = 1$$

Figura 19 - Representação da ordem de processamento dos Jobs de duas soluções.

3.5.3 Analisador de agrupamentos

O agrupamento dos *clusters* é realizado para detectar áreas promissoras no espaço de busca. É interessante descobrir essas áreas, pois se constatarmos que ela é eficiente, podemos aplicar uma busca local, explorando mais ainda uma solução do centro do *cluster*, e se percebemos que ela não é eficiente, pode-se aplicar uma perturbação com o intuito de escaparmos de um ótimo local (OLIVEIRA *et al*., 2013).

Para saber se um cluster é uma área promissora, é verificado se o seu volume chegou a um parâmetro definido empiricamente, λ, se sim, a região é considerada promissora e então o volume do *cluster* é zerado e além disso é realizada uma busca local no centro do *cluster* C_i. Caso o *cluster* tenha essa busca local aplicada r_{max} vezes seguidas sem obter melhora, será aplicada uma perturbação no centro desse *cluster*. Esta estratégia objetiva escapar de ótimos locais.

3.5.4 Algoritmo de otimização local

Logo após a CS identificar as áreas promissoras, é realizada a busca local, com o intuito de refinar a solução. Essa busca local auxilia na conversão rápida para um ótimo local (OCHOA, VEREL e TOMASSINI, 2010; HANSEN e MLADENOVI´C, 2006).

Como busca local foi implementado o algoritmo melhor melhora (HANSEN; MLADENOVIC, 2006). Recebendo uma solução por parâmetro o algoritmo melhor melhora gera todos os vizinhos da solução, e logo após, verifica qual destas soluções apresenta o melhor da função objetivo (FO). Se a FO foi melhorada, a solução com a melhor FO toma o seu lugar, e o passo anterior é repetido até que não haja mais melhora (OCHOA, VEREL e TOMASSINI, 2010).

3.5.5 Algoritmo CS

Primeiramente é criado e preenchido todos os *clusters* C_i, com uma solução gerada pela função construtiva aleatória. O volume v_i é alterado para 1 e o índice de ineficácia r_i tem o valor igual a zero. Em seguida o algoritmo SA é iniciado, realizando as iterações até que a temperatura seja reduzida à temperatura de congelamento. Ao finalizar essa etapa, inicia-se o agrupamento interativo, conforme o passo 26 da Figura 20**Erro! Fonte de referência não encontrada.**.

Nesta etapa é calculada a distância de todos os *clusters* com a solução corrente s gerada pelo SA, o *cluster* com menor distância é selecionado e o seu volume é incrementando. Se a solução do centro tiver o valor da FO inferior a *s*, então o centro de C_i é atualizado e inicia-se a etapa do analisador de agrupamentos. Neste momento é verificado se o volume v_i igualou ao parâmetro pré-estabelecido λ. Se sim, a região é considerada promissora e então o volume do *cluster* é zerado. Além disso é realizada uma busca local no centro do *cluster* C_i e atribuído a s. Se a busca local obteve melhora, o índice de ineficácia volta a ser zero, e o *cluster* é atualizado com essa solução, se não, o índice de ineficácia é incrementado e é verificado se ele chegou ao valor máximo estabelecido pelo valor calibrável r_{max}. Caso sim, o centro do *cluster* é atualizado com uma nova solução criada por um algoritmo construtivo aleatório, e essas etapas são repetidas até que a temperatura do SA congele.

Posteriormente, é identificada a melhor solução compreendida nos *clusters*. A Figura 20 apresenta o pseudocódigo da CS utilizando o SA como gerador de solução.

```
1.  Criar γ novas soluções (clusters) aleatoriamente
2.  v_i ← 1 e r_i ← 0 ∀ i = 1,...,γ   s ← solução inicial   T ← T_0
3.  vetSol ← vetor_de_solucao; qtd ← 1  vetSol[qtd] ← s  best ← 0
4.  ENQUANTO T > T_C FAÇA
5.     iter ← 0
6.     ENQUANTO iter < SA_max FAÇA
7.        iter ← iter + 1     vetSol[qtd] ← N(s)                    // N = vizinho
8.        SE f(vetSol[qtd]) < f(s) ENTÃO
9.           s ← vetSol[qtd]
10.          SE f(vetSol[qtd]) < f(vetSol[best]) ENTÃO
11.             best ← qtd  qtd  ← qtd + 1
12.          FIM-SE;
13.       SENÃO
14.          Com probabilidade e^(-(f(s)-f(s'))/T)  s ← vetSol[qtd]
15.       FIM-SE
16.    FIM-ENQUANTO
17.    T ← αT     i ← arg min_{i∈{1,...,γ}} {H_i}     v_i ← v_i + 1     c_i ← melhor(s,c_i)
18.    SE v_i = λ ENTÃO
19.       v_i ← 0    s ← busca local(c_i)
20.       SE f(s) = f(c_i) ENTÃO
21.          r_i ← r_i + 1
22.          SE r_i = r_max ENTÃO
23.             r_i ← 0    c_i ← N(c_i)
24.          FIM-SE
25.       SENÃO
26.          r_i ← 0   c_i ← s
27.       FIM-SE
28.    FIM-SE
29.    SE f(soluções[best]) > f(c_i) ENTÃO
30.       soluções[qtd] ← c_i  Best = qtd  qtd = qtd + 1
31.    FIM-SE
32. FIM-ENQUANTO    Retornar vetSol
```

Figura 20 – Pseudocódigo da CS utilizando SA.

Adaptado de ARAÚJO e MAURI, (2013) para encontrar soluções multiobjetivo.

3.6 SELEÇÃO DAS SOLUÇÕES NÃO-DOMINADAS

Após o termino do algoritmo CS, é realizado a seleção das soluções não dominadas (JOZEFOWIEZ *et al.*, 2008). Para isso, o vetor é percorrido, e a cada interação realiza-se a comparação da solução percorrida s_i com as demais soluções s_j, verificando se s_i domina s_j, ou seja, se os seus dois objetivos são menores ou iguais aos objetivos de s_j, a solução s_j é removida do vetor. Caso a solução s_i seja dominada por s_j, então s_i é removida do vetor. O pseudocódigo da identificação das soluções não dominadas é descrito na Figura 21.

```
1.  ENQUANTO i < qtd FAÇA
2.      j ← i + 1;
3.      ENQUANTO j <= qtd FAÇA
4.         SE Domina(vetSol[i], vetSol[j]) ENTÃO          // vetSol[i] domina vetSol[j]
5.             vetSol[j] = vetSol[qtd]
6.             qtd ← qtd - 1
7.             j ← j - 1
8.         SENÃO
9.             SE Domina(vetSol[j], vetSol[i]) ENTÃO
10.                vetSol[i] = vetSol[qtd]
11.                qtd ← qtd - 1
12.                j ← i
13.            FIM-SE
14.        FIM-SE
15.        j ← j + 1
16.     FIM-ENQUANTO
17.     i ← i + 1
18. FIM-ENQUANTO
19. FIM.
```

Figura 21 – Pseudocódigo da seleção das soluções não dominadas proposta nesse trabalho.

4. RESULTADOS

Os experimentos foram conduzidos em um computador com processador Intel Core i5 750 de 2,67GHz, 8GB de RAM e sistema operacional Windows 7. O algoritmo foi implementado na linguagem C++.

Durante os testes realizados, foram verificadas diversas combinações entre os parâmetros do SA e da CS. Os valores foram definidos empiricamente e ajustados de acordo com os resultados obtidos nos testes. A combinação de parâmetros e seus respectivos valores estão dispostos na Tabela 1.

Tabela 1 – Combinação da calibração dos valores para os parâmetros.

SA	SA_{max}	1000	3000	5000	8000	10000	120000
	α	0,95	0,96	0,97	0,975	0,980	0,985
	T_0	100	200	400	600	800	1000
	T_C	0,001	0,0015	0,0018	0,0019	0,0011	0,0112
CS	γ	5	10	15	20	25	30
	λ	5	10	15	20	25	30
	r_{max}	1	4	6	8	10	12

As combinações realizadas com os parâmetros da SA são: SA_{max} que representa a quantidade de iterações do SA (1000; 3000; 5000; 8000; 10000; 20000), α que indica a taxa de resfriamento (0,95; 0,96; 0,97; 0,975; 0,98; 0,985), T_0 é a temperatura inicial (100; 200; 400; 600; 800; 1000), e T_C é temperatura de congelamento (0,01; 0,0015; 0,0018; 0,0019; 0,0011; 0,0012). Os parâmetros da CS são: γ indica a quantidade de *clusters* (5; 10; 15; 20; 25; 30), λ é o volume máximo para cada *cluster* (5; 10; 15; 20; 25; 30), r_{max} é o índice de ineficácia máximo para cada *cluster* (1; 4; 6; 8; 10; 12). Foram realizados testes utilizando as mesmas instâncias, o critério de parada foi definido em 180 segundos sem melhora. Ao final dos testes, a combinação de parâmetros que obteve melhores resultados é apresentada na Tabela 2.

Tabela 2 - Valores definido dos parâmetros.

SA	SA_{max}	3000
	T_0	1000
	T_C	0,0018
CS	γ	10
	λ	20
	r_{max}	4

Para cada instância o algoritmo CS foi executado 5 vezes. Para analisar a eficiência do algoritmo, na Tabela 3 foram realizadas comparações com os melhores valores obtidos na literatura considerando o *makespan*, que é descrita a seguir: Na primeira coluna lista-se os nomes das instâncias, seguido dos respectivos limitantes superiores e inferiores conhecidos; As demais colunas listam os melhores valores encontrados por alguns trabalhos encontrados na literatura, seguidas com o tempo em segundos para entrar a solução (GONZÁLEZ *et al.*, 2015; CRUZ-CHÁVEZ *et al.*, 2015) de modo a comparar com os gerados pela CS apresentado neste estudo.

Tabela 3 – Principais resultados obtidos da minimização do *makespan* na literatura.

Instâncias (j x m)	(LB,UB)	TS	hGA	SA-Partial controlled	Tempo (s)	SSPR	Tempo (s)	CS proposto	Tempo (s)
Mk01 (10 x 6)	(36,42)	**40**	**40**	**40**	3200	**40**	11	**40**	1
Mk02 (10 x 6)	(24,32)	**26**	**26**	**28**	4600	**26**	15	**26**	28
Mk03 (15 x 8)	(204,211)	**204**	**204**	216	6786	**204**	24	**204**	2
Mk04 (15 x 8)	(48,81)	**60**	**60**	**60**	5467	**60**	19	**60**	7
Mk05 (15 x 4)	(168,186)	173	172	**168**	6781	172	57	173	38
Mk06 (10 x 15)	(33,86)	58	58	59	3783	**57**	40	58	33
Mk07 (20 x 5)	(133,157)	144	**139**	147	7526	**139**	84	140	110
Mk08 (20 x 10)	523	**523**	**523**	524	8792	**523**	83	**523**	4
Mk09 (20 x 10)	(299,369)	**307**	**307**	**307**	4563	**307**	52	**307**	7
Mk10 (20 x 15)	(165,296)	198	197	197	7865	**196**	94	200	189

Ao analisarmos a Tabela 3, percebe-se que a metaheurística CS proposta neste trabalho chegou na melhor solução conhecida em seis das dez instâncias. Nas instâncias que a CS não chegou nos melhore resultados, a mesmo obteve um valor muito próximo, o que demostra a qualidade dos resultados. Ao analisar o tempo total

para encontrar a solução, o algoritmo CS foi mais de vinte vezes mais rápido para encontrar a melhor solução obtida na literatura.

De modo a realizar uma análise multi-objetiva, considerando além do objetivo clássico para o FJSP – *makespan* – o objetivo minimizar o atraso total, aplicou-se a CS para as instâncias geradas e realizou-se uma busca considerando ambos os objetivos. Dessa forma, apresenta-se na Tabela 4 e **Erro! Fonte de referência não encontrada.**, a quantidade de soluções não dominadas para cada instância, seguido pelas soluções não dominadas com os respectivos valores encontrados de *makespan* e atraso total, ordenadas respectivamente crescente e decrescente.

Tabela 4 - Soluções não dominadas das Instâncias de Brandimarte B1.

Instâncias (j x m)	qtd não dominadas	CS proposto	
		MK	AT
Mk01_B1 (10 x 6)	4	**40**	319
		41	245
		42	224
		43	218
Mk02_B1 (10 x 6)	3	**26**	167
		31	155
		32	133
Mk03_B1 (15 x 8)	3	**204**	1592
		221	1589
		223	1584
Mk04_B1 (15 x 8)	6	**60**	683
		69	602
		70	573
		72	534
		73	532
		81	518
Mk05_B1 (15 x 4)	5	173	2233
		184	1894
		187	1746
		189	1569
		191	1545

Tabela 5 - Soluções não dominadas das Instâncias de Brandimarte B1 parte 2.

Instâncias (j x m)	qtd não dominadas	CS proposto	
		MK	AT
Mk07_B1 (20 x 5)	6	140	2251
		141	2199
		142	2183
		158	1534
		162	1399
		165	1388
Mk08_B1 (20 x 10)	3	**523**	6022
		526	5976
		536	5876
Mk09_B1 (20 x 10)	9	**307**	5486
		308	5480
		320	4265
		329	4113
		330	4112
		333	4051
		344	4017
		345	4003
		346	3917
Mk10_B1 (20 x 15)	8	200	3525
		201	3517
		202	3485
		227	3183
		228	3118
		238	3081
		239	3068
		242	3060

Tabela 6 - Soluções não dominadas das Instâncias de Brandimarte B2.

Instâncias (j x m)	qtd não dominadas	CS proposto	
		MK	AT
Mk01_B2 (10 x 6)	3	**40**	210
		42	147
		43	139
Mk02_B2 (10 x 6)	3	**26**	28
		28	10
		29	9
Mk03_B2 (15 x 8)	3	**204**	1134
		207	1130
		221	1117
Mk04_B2 (15 x 8)	4	**60**	604
		61	579
		70	414
		71	378
Mk05_B2 (15 x 4)	8	173	2190
		174	2151
		176	2135
		177	2106
		182	1540
		186	1523
		187	1393
		190	1387
Mk06_B2 (10 x 15)	4	59	338
		63	311
		67	307
		68	283

Tabela 7 - Soluções não dominadas das Instâncias de Brandimarte B2.

Instâncias (j x m)	qtd não dominadas	CS proposto	
		MK	AT
Mk07_B2 (20 x 5)	6	140	1699
		142	1667
		153	976
		154	884
		156	866
		169	845
Mk08_B2 (20 x 10)	4	**523**	6066
		524	5917
		527	5894
		528	5468
Mk09_B2 (20 x 10)	4	**307**	4881
		308	4874
		323	3402
		326	3377
Mk10_B2 (20 x 15)	5	202	2866
		229	2532
		233	2498
		236	2468
		237	2348

Com a Instância Mk10_B1, foi gerado o gráfico de Pareto, disponibilizando as soluções não dominadas para o tomador de decisão ter mais informações para assim ter um veredito mais confiável.

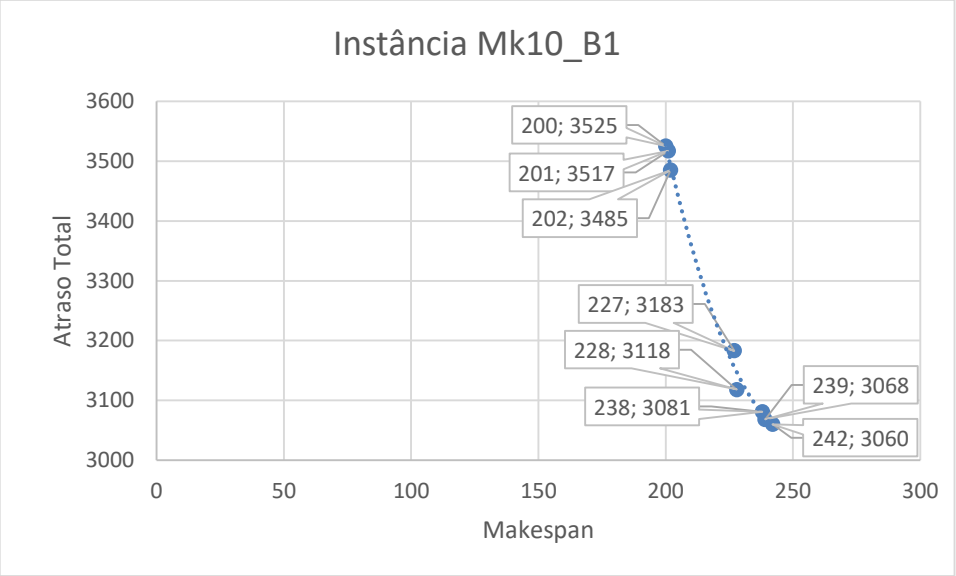

Figura 22 – Curva de Pareto da instancia Mk10_B1.

5. CONCLUSÕES

O presente trabalho abordou o *Job Shop Scheduling* Flexível Bi-Objetivo. O modelo utilizado para este problema é baseado no modelo proposto por Melo e Ronconi, (2015) e adaptado para considerar também o *makespan* como objetivo.

Para a resolução do problema proposto foi utilizada uma adaptação meta-heurística *Clustering Search* usando a *Simulated Annealing* como geradora de soluções para considerar problemas multiobjetivos.

Os resultados obtidos pela CS analisando o *makespan* se mostraram satisfatórios em relação a literatura, encontrando a solução ótima em 6 de 10 problemas clássicos. Com isso a CS se mostrou como uma boa alternativa para a resolução do FJSP bi-objetivo, encontrando várias soluções não-dominadas para o problema.

6. REFERÊNCIAS

ALIANO FILHO, A. **Metaheurísticas em um problema de rotação de culturas**, Dissertação de Mestrado em Biometria, Universidade Estadual Paulista Júlio de Mesquita Filho – IBB – UNESP, 2012 (www.ibb.unesp.br/posgrad/teses/biometria_me_2012_angelo_filho.pdf), 5, 2012.

ARAÚJO, D. L.; MAURI, G. R. Clustering search para resolução de um problema de rotação de culturas com restrições de adjacências. In: **Simpósio Brasileiro de Pesquisa Operacional**, Natal, v. 11, p. 610-621, 2013.

ARENALES, M.; ARMENTANO, V. A.; MORABITO, R.; YANASSE, H.H. **Pesquisa operacional**. 6 ed. Rio de Janeiro, Elsevier, p. 524, 2007.

BRANDIMARTE, P. Routing and scheduling in a flexible job shop by tabu search. **Annals of Operations Research**, Torino, v. 41, p 157–183, 1993.

BISSOLI, D. C.; AMARAL, A. R. S. Um GRASP Simples e Robusto para o Job Shop Scheduling Problem. In: XLVII CONGRESSO BRASILEIRO DE PESQUISA OPERACIONAL, Pernambuco, v. 10, p. 2057-2067 2015. Disponível em: http://www.sbpo2015.iltc.br/pdf/142810.pdf. Acesso em: 10 out. 2016.

CARVALHO, L C, F. ALGORITMO HÍBRIDO MULTIOBJETIVO PARA O PROBLEMA FLEXIBLE JOB SHOP. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Uberlândia, Minas Gerais, 2015.

CARVALHO, M. H.; CERIOLI, M.R.; DAHAB, R.; FEOFILOFF, P.; FERNANDES, C. G.; FERREIRA, C. E.; GUIMARÃES, K. S.; MIYAZAWA, F. K.; PINA JR, J. C.; SOARES, J. A. R.; WAKABAYASHI, Y. **Uma introdução sucinta a algoritmos de aproximação**. 23º Colóquio Brasileiro de Matemática: [s.n.], p. 157, 2001.

CHAUDHRY, I. A.; KHAN, A. A. A research survey: review of flexible job shop scheduling techniques. **International transactions in operational research**. Islamabad, Pakistan, v. 23, n. 3, p. 551–591, 2016.

CHIAVENATO, I. **Introdução à Teoria Geral da Administração**. Campos. 7 ed. S. Paulo: McGraw-Hill do Brasil, p. 634, 2004.

CRUZ-CHÁVEZ, M. A.; MARTÍNEZ-RANGEL, M. G.; CRUZ-ROSALES, M. H. Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem. **International Transactions in Operational Research**. v. 24, p.1119–1137, 2017.

GOMES JÚNIOR, A. DE C. **Problema de Sequenciamento em uma Máquina com Penalidades por Antecipação e Atraso: Modelagem e Resolução.** 2007. 73 f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Minas Gerais, Belo Horizonte.

GONZÁLEZ, M. A.; VELA, C. R.; VARELA, R. Scatter search with path relinking for the flexible job shop scheduling problem. **European Journal of Operational Research**. v. 245, n. 1, p. 35–45, 2015.

HAESER, G.; GOMES–RUGGIERO, M. Aspectos Teóricos de Simulated Annealing e um Algoritmo duas Fases em Otimização Global. **Sociedade Brasileira de Matemática Aplicada e Computacional**, v. 9, n. 3, p. 395-404, 2006.

HAMMING, R. W. Error detecting and error correcting codes. **Bell System Technical Journal**, v. 26, p. 147-160, 1950.

HANSEN, P.; MLADENOVI´C, N. First vs. best improvement: An empirical study. **Discrete Applied Mathematics**. v. 154, p. 802–817, 2006.

HO, N. B.; TAY, C. J. GENACE: an effective chromosome representation for evolving flexible job shop schedules. **European Journal of Operational Research**. v. 2, p. 1759 – 1766, 2004.

JOZEFOWIEZ, N., SEMET, F., TALBI, E.-G.: **From single-objective to multiobjective vehicle routing problems: Motivations, case studies, and methods**. In: Golden, B.L., Raghavan, S., Wasil, E. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges, p. 445–471, 2008.

KIRKPATRICK, S.; GELATT, C.; VECCHI, M. Optimization by simulated annealing. **Science**, v. 220, n. 4598, p. 498-516, 1983.

KOULAMAS, C. The Total Tardiness Problem: Review and Extensions. **Operations Research**, p. 1025-1041, 1994.

LAGUNA, M. and MARTÍ, R. Grasp and path relinking for 2-layer straight line crossing minimization. **INFORMS Journal on Computing**, Valencia, 11:44–52, 1999.

MAURI, G. R. Novas heurísticas para o problema de escalonamento de tripulações. Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisa Espacial, São José dos Campos, 2005.

MELO, E, L.; RONCONI, D. Regras de prioridade eficientes que exploram características do Job Shop Flexível para a minimização do atraso total. **Production**, v. 25, n. 1, p. 79-91, 2015.

MOREIRA, M. C. de O. *et al*. Um algoritmo evolutivo para o problema de job shop scheduling bi-objetivo Problem. In: XL Congresso brasileiro de pesquisa operacional, João pessoa, PB, p. 1446-1457, 2008**.**

OCHOA, G.; VEREL, S.; TOMASSINI, M. First-improvement vs. Best-improvement Local Optima Networks of NK Landscapes. **ArXiv e-prints**. Krakow, Poland, v. 10, p. 104-113, 2010.

OLIVEIRA, A, C. M.; CHAVES, A. A; LORENA, L. A N. Clustering Search. **Pesquisa Operacional**. v. 33, n. 1, p.105-121, 2013.

RABELLO, R. L.; MAURI, G. R.; RIBEIRO, G. M.; LORENA, L. A. N. Meta-heurística clustering search aplicada ao problema de rotulação cartográfica de pontos. Simpósio Brasileiro de Pesquisa Operacional, Natal, p. 3358-3365, 2013.

RABIEE, M.; ZANDIEH, M.; RAMEZANI, P. Bi-objective partial flexible job shop scheduling problem: NSGA-II, NRGA, MOGA and PAES approaches, **International Journal of Production Research**. p. 7327-7342, 2012.

RIBEIRO FILHO, G.; NAGANO, M. S.; LORENA, L. A. N. Evolutionary Clustering Search for Flowtime Minimization in Permutation Flow Shop. **Lecture Notes in Computer Science**. v. 4771 p. 69-81, 2006.

Rodammer, F. A.; White, K. P. A Recent Survey of Production Scheduling. **IEEE Transactions on Systems, Man and Cybernetics**. v. 18, n. 11, p. 841-851, 1988.

SCRICH, C. R. **BUSCA TABU PARA A PROGRAMAÇÃO DE TAREFAS EM JOB SHOP COM DATAS DE ENTREGA**. 117 f. Tese (Doutorado em Engenharia Elétrica)

– Programa de Pós-Graduação em Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, São Paulo, 1997.

SIERKSMA, G. **Linear and integer programming**: Theory and Practice. MARCEL DEKKER, INC, v. 673 p. 37, 1996.

SINGH, M. R.; MAHAPATRA, S. S. A quantum behaved particle swarm optimization for flexible job shop scheduling. **Computers & Industrial Engineering**, v. 93, p. 36-44, 2016.

Yuan, Y.; Xu, H. Multiobjective Flexible Job Shop SchedulingUsing Memetic Algorithms. **IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING**. v. 12, n. 1, p.336-352, 2015.