UNISINOS

Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Leandro Mengue da Silva

Um Modelo de Otimização Baseado em Algoritmo Memético Para o Escalonamento de Ordens de Produção Utilizando Divisão de Lotes de Tamanho Variável

São Leopoldo, 2017

UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO
EM COMPUTAÇÃO APLICADA
NÍVEL MESTRADO

LEANDRO MENGUE DA SILVA

UM MODELO DE OTIMIZAÇÃO BASEADO EM ALGORITMO MEMÉTICO PARA O ESCALONAMENTO DE ORDENS DE PRODUÇÃO UTILIZANDO DIVISÃO DE LOTES DE TAMANHO VARIÁVEL

São Leopoldo
2017

S586m Silva, Leandro Mengue da.

Um modelo de otimização baseado em algoritmo memético para o escalonamento de ordens de produção utilizando divisão de lotes de tamanho variável / Leandro Mengue da Silva. – 2017.

91 f. : il. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, 2017.

"Orientador: Dr. Arthur Tórgo Gómez."

1. Algoritmo memético. 2. Otimização. 3. Lotes variáveis. 4. Escalonamento. 5. Divisão de lotes. I. Título.

CDU 004

Dados Internacionais de Catalogação na Publicação (CIP)

(Bibliotecário: Flávio Nunes – CRB 10/1298)

Leandro Mengue da Silva

Um Modelo de Otimização Baseado em Algoritmo Memético
Para o Escalonamento de Ordens de Produção Utilizando
Divisão de Lotes de Tamanho Variável

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em ____ de _____ de _____.

BANCA EXAMINADORA

Prof. Dr. Arthur Tórgo Gómez - UNISINOS

Prof. Dr. José Vicente Canto dos Santos - UNISINOS

Prof. Dr. Leonardo Dagnino Chiwiacowsky - UCS

Prof. Dr. Arthur Tórgo Gómez (Orientador)

Visto e permitida a impressão

São Leopoldo,

Prof. Dr. Sandro José Rigo
Coordenador PPG em Computação Aplicada

Este trabalho é dedicado à minha família.

AGRADECIMENTOS

A minha esposa Silvia e a meus filhos Lucas e Mateus pelo amor, carinho e apoio incondicional.

Ao Prof. Dr. Arthur Tórgo Gómez pelos ensinamentos e orientações neste período.

A todos os professores do PIPCA-UNISINOS pelos ensinamentos, em especial aos professores José Vicente, Leonardo Chiwiacowsky e Luiz Paulo Luna.

Ao colega e amigo Márcio Barth pela parceria nos trabalhos das disciplinas e artigos.

Aos amigos Larri Laux, Lucas Poglia e Ana Paula Pagnoncelli pelas colaborações ao trabalho.

Aos amigos Ricardo Hoffmann e Marcelo Antunes por acreditarem em nosso projeto de otimizador de produção.

Aos professores Roberto da Silva, Acirete Simões, Sandra Prado, Rosandra Mottola, Vinicius Brei e Felipe Marques pelas recomendações.

Ao CNPQ pela bolsa de estudos que possibilitou minha participação no Mestrado em Computação Aplicada.

A Deus pelas oportunidades e por colocar todas essas pessoas em minha vida.

RESUMO

A contribuição de metaheurísticas, em especial a dos algoritmos evolutivos, na área de otimização combinatória é de extrema relevância, pois auxiliam na busca de soluções próximas ao ótimo para problemas complexos da vida real cuja resolução em tempo aceitável é inviável devido a sua complexidade computacional, oferecendo uma flexibilidade importante na modelagem do problema. Este trabalho se propõe a apresentar e implementar um modelo computacional a ser utilizado na otimização do escalonamento de ordens de produção utilizando um Algoritmo Memético (AM), que permite a busca tanto da melhor sequência das ordens de produção quanto dos lotes de tamanho variável em que a quantidade de cada operação pode ser subdividida. A possibilidade de utilização de máquinas alternativas, de recursos secundários, de intervalos de indisponibilidade e de lotes de transferência, é apresentada no modelo, o que lhe proporciona grande robustez e aplicabilidade em ambientes de manufatura flexível, permitindo uma modelagem do *Flexible Job Shop Scheduling Problem* (FJSSP) que reflete com maior fidedignidade a realidade do ambiente fabril, gerando como resultado um escalonamento otimizado e aderente às necessidades da fábrica. Várias instâncias do FJSSP são utilizadas nos testes e os resultados obtidos comprovam que o algoritmo proposto consegue otimizar o escalonamento das ordens de produção de cada instância de maneira eficiente.

PALAVARAS CHAVE: Algoritmo Memético, Otimização, Lotes Variáveis, Escalonamento, Divisão de Lotes.

ABSTRACT

The contribution of meta-heuristics, especially evolutionary algorithms, in combinatorial optimization area is extremely important, as they help in finding near optimal solutions to complex real-life problems whose resolution is infeasible in acceptable time due to its computational complexity, offering an important flexibility in the modeling of problem. This study propose to present and implement a computational model to be used in optimizing the production scheduling of manufacturing orders using a Memetic Algorithm that allows to search both the best sequence of jobs as of variable size batches that the quantity of each operation can be subdivided. The possibility of using alternative resources, operations with secondary resources, unavailability intervals and batch transfer lots are features presented in the model, which lends it great robustness and applicability to flexible manufacturing environments, allowing the modeling of Flexible Job Shop Scheduling Problem (FJSSP) that reflects with higher accuracy the real manufacturing environment, generating optimized scheduling results that are adhering to the plant needs. Multiple instances of FJSSP are used in the tests and the results show that the proposed algorithm succeeds in optimizing the scheduling of production orders for each instance so efficient.

Keywords: Memetic Algorithm, Optimization, Variable Lot Size, Flexible Job Shop, Lot Stream, Lot Split.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AG	Algoritmo Genético
AM	Algoritmo Memético
BT	Busca Tabu
CPM	*Critical Path Method*
FJSSP	*Flexible Job Shop Scheduling Problem*
FJSSP-LS	*Flexible Job Shop Scheduling Problem with Lot Streaming*
JSSP	*Job Shop Scheduling Problem*
LB	*Lower Bound*
LS	*Lot Streaming*
PIPCA	Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
PMX	*Partially Mapped Crossover*
SMF	Sistema de Manufatura Flexível

SUMÁRIO

1 INTRODUÇÃO

Os conceitos de escalonamento, sequenciamento e programação não são novos. Sun Tzu escreveu sobre sequenciamento e estratégia em 500 A.C. sob uma perspectiva militar, pirâmides foram edificadas a 3000 anos atrás, rodovias transcontinentais foram construídas durante 200 anos. Nenhuma destas atividades poderia ter sido realizada sem alguma forma de programação, ou seja, a compreensão das atividades e do sequenciamento (WEAVER, 2006). No final da década de 1950, os trabalhos de Walker e Kelley sobre *Critical Path Method* (CPM) estabeleceram definitivamente o problema de sequenciamento, sendo abordado por meio de um método científico. Desde então, muitas técnicas e modelos foram elaborados no sentido de resolver problemas de escalonamento, sequenciamento e programação de atividades com recursos limitados.

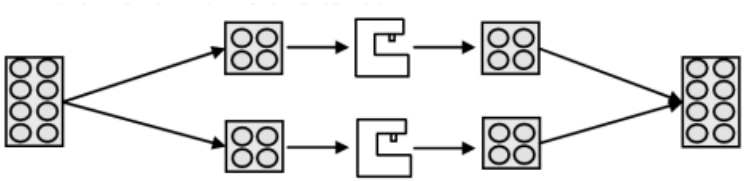
O escalonamento de atividades é utilizado nas mais diversas áreas. Projetos de toda natureza são planejados programando-se as tarefas a serem executadas temporalmente e quais recursos serão necessários para suas execuções. Em algumas, áreas o processo de planejamento é tão recorrente que é executado diariamente ou semanalmente com a finalidade de refletir no plano as condições em que as tarefas já realizadas foram executadas, adequando, se necessário, as diferenças entre o cenário planejado e o realizado. Este é exatamente o contexto do escalonamento das atividades de uma fábrica, onde deseja-se, a partir de uma condição inicial, alocar da melhor maneira as tarefas relativas às operações necessárias para produzir os produtos com uso dos recursos disponíveis (máquinas, pessoas, ferramentas, etc.). A esse cenário dá-se o nome de *Job Shop* e ao Problema de Escalonamento das Ordens de Produção denominamos de *Job Shop Scheduling Problem* (*JSSP*).

O JSSP é um dos problemas mais importantes em um Sistema de Manufatura Flexível (SMF), pois a alocação temporal adequada dos recursos é fundamental para o aumento da eficiência da produção (VARGA, 2014), e é por esse motivo que ele é extensamente estudado. Porém, as pesquisas relativas ao JSSP, na maioria das vezes, desconsideram importantes fatores do SMF, resultando em modelos mais simplificados e que não refletem a realidade de um ambiente fabril. Nesse sentido, uma subclasse do JSSP foi criada para refletir a flexibilidade existente no SMF, o *Flexible Job Shop Scheduling Problem* (FJSSP), onde são consideradas situações importantes como a utilização de máquinas alternativas para uma mesma operação.

O FJSSP trouxe novas possibilidades em relação ao escalonamento no SMF, entretanto, ainda assim, os modelos não conseguiam refletir a realidade da fábrica devido a simplificações ainda adotadas nos modelos (PENHA et al., 2014). Uma das simplificações mais importantes e frequentemente utilizadas é a desconsideração da divisão de lotes (*LS – lot streaming* ou *lot split*), que retira do problema uma flexibilidade importante do SMF (YOON; VENTURA, 2002$_a$), onde é muito comum encontrar situações em que, objetivando o aumento de produtividade, existem várias máquinas que podem executar a mesma operação e, desse modo, o que se deseja é que um lote de produção seja dividido para ser processado em paralelo nessas máquinas alternativas.

A Figura 1 apresenta um exemplo onde se observa que, mesmo em sistemas bastantes simples como este com duas máquinas, há a necessidade de divisão do lote de produção para que o processo de produção possa ocorrer em paralelo em mais de uma máquina.

Figura 1: Divisão de lotes



Fonte: KOIKE et al., 2008

Devido à importância do escalonamento, utilizando divisão de lotes, é proposto neste trabalho um modelo onde o *Flexible Job Shop Scheduling Problem with Lot Stream* (FJSSP-LS) é estabelecido e otimizado através de um AM que utiliza a divisão de lotes de tamanho variável em seu modelo. De forma mais específica, o modelo se propõe a:

- Possibilitar a divisão do lote de produção em lotes menores com tamanhos que podem variar para cada operação da ordem de produção;
- Suportar características importantes como a utilização de máquinas alternativas, a necessidade de recursos secundários e a consideração de intervalos de indisponibilidade, onde turnos, fins de semana e paradas de manutenção podem ser mapeados;
- Prover uma melhora no escalonamento de produção, através da utilização de lotes de transferência e do método de prevenção de falta de fluxo;
- Possibilitar a otimização de cenários de escalonamento de produção, gerando resultados mais eficientes e aderentes à realidade da fábrica.

Este trabalho está dividido em 7 capítulos onde, após esta introdução, são abordados, no capítulo 2, os aspectos mais relevantes do problema de escalonamento, como conceitos, definições, classificação computacional, importância no SMF, publicações, principais elementos que o compõem e três dos principais métodos de resolução do JSSP, através dos métodos *Branch-and-Bound*, Busca Tabu e Algoritmo Genético (AG). No capítulo 3, é apresentado o modelo proposto neste trabalho para a resolução do FJSSP-LS, baseado no AM. No capítulo 4, são discutidos os principais aspectos da implementação do modelo proposto, como linguagem, estruturas e paralelismo. No capítulo 5, são avaliados casos de testes unitários e de escalonamentos clássicos e com divisão de lotes. No capítulo 6 são apresentados os resultados obtidos. No capítulo 7, são apresentadas as conclusões finais sobre o trabalho e as propostas de trabalhos futuros.

2 O PROBLEMA DE ESCALONAMENTO

O Problema de Escalonamento de Ordens de Produção é um problema que tem origem no planejamento da produção, onde se deseja alocar os recursos produtivos (como máquinas e pessoas) a fim de permitir o processamento das ordens de produção, respeitando as restrições de capacidade desses recursos, bem como obedecendo a precedência entre as operações necessárias para produzir cada ordem de produção. Esses elementos levam o JSSP a ser um problema de difícil solução, pois muitas alternativas (combinações) são possíveis para a resolução do problema, dificultando a obtenção de um escalonamento ótimo (SIMON, 2014).

O JSSP tem sido extensamente estudado por pesquisadores devido a sua relevância no processo produtivo desde o início da revolução industrial (GEIRSSON, 2012). Os ambientes produtivos tornam-se cada vez mais complexos e a utilização de muitos recursos produtivos necessita que as tarefas, a serem executadas em cada máquina, tenham uma sequência adequada a fim de maximizar a utilização destes recursos produtivos. Nesse contexto, criou-se o conceito de SMF, onde vários elementos como máquinas e pessoas se integram com o objetivo de proporcionar um ambiente flexível que possa adaptar-se às mudanças no cenário produtivo com maior facilidade. Considerando-se o ambiente, o SMF pode ser tipificado de acordo com suas características:

- ***Flow Shop***: neste tipo de ambiente, as peças seguem um fluxo sempre com a mesma rota, ou seja, todos os produtos têm o mesmo roteiro de fabricação no que diz respeito à sequência de máquinas que as operações devem seguir;
- ***Job Shop***: ambiente onde cada tipo de peça possui seu próprio roteiro de fabricação, onde o número de operações e a máquina que cada uma requer podem ser diferentes para diferentes tipos de peças;
- ***Open Shop***: tipo de ambiente onde todas as peças têm o mesmo roteiro de fabricação e utilizam todas as máquinas, porém a ordem das operações executadas nas máquinas não tem restrição, ou seja, podem ser executadas em qualquer sequência, podendo peças diferentes ter sequenciamento diferente.

2.1 Conceitos básicos do problema de escalonamento

Neste trabalho, alguns conceitos fundamentais do JSSP serão utilizados com frequência, e assim seu entendimento é fundamental no desenvolvimento das ideias que serão propostas. Dessa forma, são descritos a seguir alguns conceitos importantes no âmbito do JSSP:

- **Ordem de produção**: documento que contém informações de produção, como por exemplo, produto, quantidade e data de entrega desejada;
- **Operação**: processo a ser executado em uma peça e que contribui parcialmente na produção do produto;
- **Roteiro de fabricação**: sequência de operações necessárias para produzir um determinado produto;

- **Máquina/ferramenta/pessoa**: recurso do SMF necessário para a execução de uma operação;

- ***Setup***: tempo gasto em todas as atividades de preparação de uma máquina para ficar apta a executar uma determinada operação;

- **Data desejada de entrega**: data em que se deseja que uma ordem de produção esteja completamente finalizada;

- **Data de entrega da ordem**: data da finalização da última operação da ordem de produção;

- ***Makespan***: tempo total de produção de um cenário de escalonamento, calculado como o tempo decorrido entre o início do escalonamento das ordens de produção e o final da última operação da ordem escalonada mais tardiamente.

2.2 Definição do problema clássico de escalonamento

Desde os primeiros trabalhos dedicados a definir e resolver o JSSP (BOWMAN, 1959), (GRAHAM et al., 1979), muitos autores trabalharam o assunto do escalonamento de produção, resultando que o JSSP é um problema claramente estabelecido, podendo ser definido da seguinte forma (NOWICKI; SMUTNICKI, 1996): seja um conjunto de ordens de produção (*jobs*) $J = \{1, ..., n\}$ em que cada ordem J_i possui um conjunto de operações $O = \{1, ..., o\}$ que devem ser processadas em sequência pré-estabelecida em uma das máquinas do conjunto $M = \{1, ..., m\}$, sendo i o índice da ordem de produção, j o índice da operação e l o índice da máquina, com tempo de processamento T_{ijl} pré-estabelecido. Adicionalmente, deve-se considerar que as máquinas só podem processar uma peça por vez, impedindo sobreposições de tarefas numa mesma máquina. Assim, o objetivo do problema é buscar a melhor configuração da sequência de execução das ordens de produção, de forma a maximizar (ou minimizar, dependendo do caso) o valor da função objetivo.

2.3 Complexidade computacional do problema de escalonamento

A complexidade computacional de um algoritmo, utilizado para resolver um problema, está ligada à relação entre o tempo necessário para sua execução e o tamanho da instância do problema. Assim, dizemos que a complexidade de um algoritmo é, por exemplo, $O(n^2)$, caso o tempo necessário para que seja executado dependa do quadrado da variável n.

O problema de escalonamento clássico tem sua complexidade computacional basicamente determinada pelo número de ordens de produção e pelo número de máquinas disponíveis. Já no FJSSP-LS, a complexidade computacional é influenciada também pela quantidade de operações das ordens de produção, pela quantidade de máquinas alternativas de cada operação e pelo número de lotes que cada operação pode ser dividida. A complexidade computacional do algoritmo proposto é calculada e apresentada na seção 5.3.

2.3.1 Classificação em relação ao tipo de resposta

Um algoritmo pode ser classificado em três tipos de problemas, estando essa classificação ligada ao tipo de resposta do algoritmo (TOSCANI; VELOSO, 2008):

- **Problemas de decisão**: problema onde a resposta possível é sim ou não, onde um conjunto de variáveis está disposto de forma que compõe uma expressão lógica que representa o problema. Se existirem valores para as variáveis que podem tornar a expressão verdadeira, então a resposta ao problema é sim. Caso contrário é não;

- **Problemas de localização**: para um problema onde há um (ou mais) conjunto(s) de variáveis que satisfaça-o como verdadeiro, o problema de localização é aquele que busca quais os valores para essas variáveis que possibilitam ao problema ser verdadeiro;

- **Problemas de otimização**: problema onde busca-se o conjunto de variáveis que permita obter uma resposta maximizada ou minimizada do problema. O FJSSP-LS aqui estudado é um problema de otimização, que busca minimizar a função objetivo.

2.3.2 Classificação em relação à complexidade

Um algoritmo pode ser classificado em três classes de problemas, estando essa classificação intimamente ligada à sua complexidade computacional (GAREY; JOHNSON; STOCKMEYER, 1974):

- **Classe P**: são problemas possíveis de serem resolvidos em tempo polinomial por um algoritmo determinístico;

- **Classe NP**: são problemas possíveis de serem resolvidos em tempo polinomial por um algoritmo não-determinístico;

- **Classe NP-Difícil**: são problemas onde a complexidade é muito alta e que não possuem solução em tempo polinomial. Todos os três problemas de escalonamento citados (JSSP, FSSP e o FJSSP-LS) são problemas da classe NP-Difíceis (GAREY, 1976).

2.4 O problema de escalonamento de lotes e sua importância no SMF

Os SMFs apresentam características que permitem responder mais rapidamente a mudanças no cenário produtivo, adaptando-se a novas situações com maior facilidade.

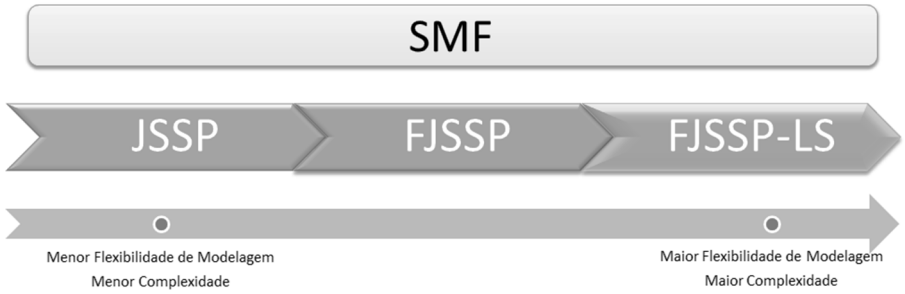
Um dos fatores mais importantes de um SMF é sua capacidade de processar peças de uma mesma operação em várias máquinas alternativas, fazendo com que o JSSP se torne um FJSSP, tendo este sido inicialmente apresentado por (BRUCKER; SCHLIE, 1990). Para possibilitar o processamento das peças de uma operação em várias máquinas, é necessário que as peças sejam divididas em partições denominadas lotes.

No contexto do FJSSP, a utilização dos recursos de forma otimizada e a busca por sequenciamentos que melhorem os índices de desempenho (como redução do atraso médio e redução dos tempos de *setup*) são objetivos constantes no planejamento de produção. Assim, para possibilitar a modelagem de sistemas que busquem soluções no contexto da manufatura flexível, a utilização de divisão de lotes se torna necessária, pois dessa maneira é possível distribuir a carga de processamento em máquinas alternativas, buscando-se otimizar os índices de desempenho do sistema.

Desse modo, temos uma nova subdivisão do problema de escalonamento: o FJSSP-LS, que é o problema de escalonamento onde a quantidade de um lote pode ser subdividida em lotes menores para que seja possível o processamento simultâneo desses lotes em máquinas alternativas. Dessa forma, o tempo de produção da quantidade total de uma operação é reduzido em relação direta com o número de máquinas disponíveis para processá-la. Por exemplo, um lote único que possui 100 peças e requer 10 minutos para produzir cada peça, levará 1000 minutos para ser concluído, mas caso haja a disponibilidade de duas máquinas para processar a operação, o lote único pode ser dividido em dois lotes onde cada um terá 50 peças e requererá 500 minutos de tempo de cada máquina. Como as máquinas estão trabalhando em paralelo, o tempo de entrega da operação será de 500 minutos, que é uma grande redução de tempo se comparada ao tempo do lote único de 1000 minutos.

A Figura 2 apresenta as três subdivisões dos problemas de escalonamento e como a complexidade e a flexibilidade crescem com o tipo de problema.

Figura 2: Subdivisões de problemas de escalonamento no SMF



Fonte: Elaborado pelo autor.

O problema de otimização de um FJSSP-LS é mais amplo do que apenas dividir a quantidade de peças de uma operação no maior número de lotes possível, de acordo com o número de máquinas paralelas, pois várias operações de várias ordens de produção podem utilizar as mesmas máquinas. Então, quando se utiliza uma divisão com um maior número de lotes, mais máquinas são ocupadas em paralelo e assim outras operações terão que aguardar mais tempo para iniciar seu processamento, o que é um efeito colateral indesejado, pois implicará no retardo da finalização dessas outras ordens de produção.

O escalonamento ideal deve balancear o número de lotes e suas quantidades a fim de possibilitar tanto o bom uso das máquinas alternativas, que possibilitam a execução de uma mesma operação em máquinas paralelas, quanto não penalizar com atrasos em demasia outras operações que necessitem de processamento nas mesmas máquinas. Este balanceamento ideal pode ser obtido através de técnicas de otimização como as que iremos apresentar.

Segundo SARIN e JAIPRAKASH (2007), a divisão de lotes pode ser feita de três formas: lotes de mesmo tamanho, lotes de tamanho consistente e lotes de tamanho variável. Nas seções seguintes será detalhado cada forma da divisão de lotes.

Figura 3: Formas de divisão de lotes

Fonte: Elaborado pelo autor.

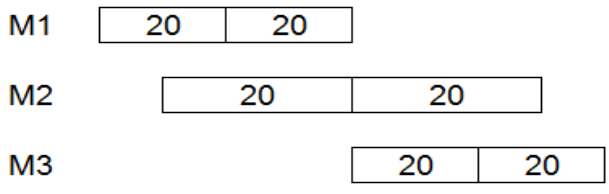
A Figura 3 apresenta as três formas de divisões de lotes no FJSSP-LS e como a complexidade e a flexibilidade crescem com a forma de divisão adotada.

2.4.1 Escalonamento com lotes de mesmo tamanho

A divisão de lotes de mesmo tamanho é a variante mais simplificada da divisão de lotes, pois necessita-se apenas saber o número de divisões que será aplicado na ordem de produção em cada operação. Esse número vale para toda a ordem de produção, ou seja, todas as operações desta ordem terão sua quantidade dividida no mesmo número de lotes. Como consequência, os tamanhos de todos os lotes de uma ordem de produção (em qualquer operação) terão a mesma quantidade.

O espaço de busca no que tange às possibilidades é bem mais restrito do que nos outros tipos de divisões de lotes (consistente e variável), pois necessitamos apenas N variáveis, cada uma representando o número de divisões D_i de cada ordem de produção, onde N é o número de ordens de produção e $D_i = [1,...,NB]$, sendo NB um parâmetro do sistema que representa o número máximo de divisões permitidas.

Figura 4: Lotes de mesmo tamanho



Fonte: Adaptado pelo autor de (SARIN; JAIPRAKASH, 2007).

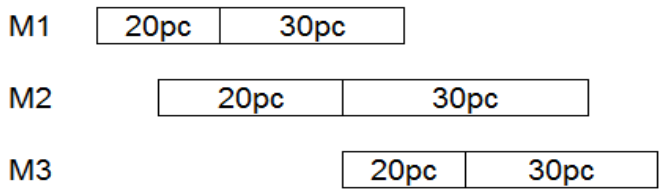
A Figura 4 mostra um exemplo de uma ordem de produção com 40 peças que foi dividida em lotes iguais. O número de divisões aplicado a todas as operações foi 2, logo cada lote tem o tamanho igual a 20 peças. Na figura, M1, M2 e M3 representam 3 máquinas distintas, onde em cada uma é processada uma operação diferente.

2.4.2 Escalonamento com lotes de tamanho consistente

A divisão de lotes de tamanho consistente permite um pouco mais de flexibilidade na divisão de lotes do que a divisão em tamanhos iguais, pois permite que os lotes formados tenham diferentes tamanhos. Porém, sua denominação “consistente” significa que todas as operações seguem o mesmo padrão de divisão, ou seja, todo k-ésimo lote de cada operação de uma ordem de produção tem o mesmo tamanho.

Em termos computacionais, o espaço de busca aumenta em relação à divisão de lotes de mesmo tamanho, pois cada ordem necessita *NB* variáveis que podem assumir qualquer valor entre zero e o total da quantidade da ordem, respeitando unicamente que a soma da quantidade dos lotes (em cada operação) seja igual à quantidade da ordem.

Figura 5: Lotes de tamanho consistente



Fonte: Adaptado pelo autor de (SARIN; JAIPRAKASH, 2007).

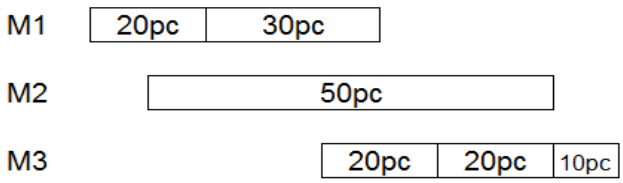
A Figura 5 mostra um exemplo de uma ordem de produção com 50 peças que foi dividida em lotes consistentes. O número de divisões aplicado a todas as operações foi 2, sendo que o primeiro lote de cada operação tem 20 peças e o segundo lote de cada operação tem 30 peças.

2.4.3 Escalonamento com lotes de tamanho variável

A divisão de lotes de tamanho variável é a forma mais flexível entre os tipos de divisão de lotes, pois permite que os lotes formados tenham diferentes tamanhos em diferentes operações. Como consequência disso, o número de lotes em cada operação pode variar, permitindo que se tenham diferentes configurações de divisões de lotes em diferentes operações da mesma ordem.

Em termos computacionais, o espaço de busca aumenta mais ainda em relação à divisão de lotes de tamanho consistente, pois cada ordem necessita $O_i * NB$ variáveis que podem assumir qualquer valor entre zero e o total da quantidade da ordem, respeitando unicamente que a soma dos lotes (de cada operação) seja igual à quantidade da ordem de produção. Aqui, O_i representa o número de operações da *i*-ésima ordem de produção.

Figura 6: Lotes de tamanho variável



Fonte: Adaptado pelo autor de (SARIN; JAIPRAKASH, 2007).

A Figura 6 mostra um exemplo de uma ordem de produção com 50 peças que foi dividida em lotes variáveis. É possível verificar a flexibilidade da divisão de lotes nesse caso, pois a primeira operação foi dividida em 2 lotes de 20 e 30 peças cada. Já a segunda operação não foi dividida, tendo apenas 1 lote de 50 peças. E finalmente a terceira operação foi dividida em 3 lotes de 20, 20 e 10 peças cada.

O modelo proposto permitirá a modelagem de cenários onde a divisão de lotes em tamanhos variáveis esteja presente, pois esta é a forma de divisão que permite maior flexibilidade de modelagem, representando de maneira mais adequada as restrições e necessidades do SMF.

2.5 Histórico de publicações

Ao estudar o FJSSP-LS, foi verificada uma razoável dificuldade de encontrar trabalhos relacionados que abordavam o assunto sob a ótica da divisão em lotes de tamanho variável, fato já identificado por LIU (2003), onde afirma que a maioria dos pesquisadores nesta área concentraram-se nos modelos de divisão de lotes consistentes.

Através da base de dados em CAPES (2016), foi possível pesquisar alguns trabalhos relacionados à divisão de lotes *(Lot Streaming* e *Lot Split*) e analisá-los, a fim de verificar se a suposição de que a divisão de lotes em tamanhos variáveis é realmente pouco explorada na literatura está correta. A Tabela 1 apresenta os trabalhos pesquisados.

Tabela 1: Trabalhos pesquisados

Trabalhos pesquisados	Enquadramento
(KHALILI; NADERI, 2015), (ROOBEEK, 1997), (YURTKURAN; EMEL, 2014), (DRISS et al., 2015), (OZCAN; TOKLU, 2009), (MOHTASHAMI, 2014), (CALIS; BULKAN, 2015), (GÓMEZ; ANDRÉS; ROMANO, 2013), (MOKHTARI, 2014), (CHO et al., 2014), (DOUSTHAGHIA; MOGHADDAMB, 2012), (CUI; GU, 2015), (ELHAFSI, 2000), (ROUNDY et al., 2005), (HORNG; HU; CHENG, 2009), (QING; WANG, 2013), (BANHARNSAKUN; SIRINAOVAKUL; ACHALAKUL, 2012),	Trabalhos que fazem alguma referência à divisão de lotes mas não abordam o FJSSP-LS no artigo. (30)

(SANGSAWANG et al., 2015), (WANGA et al., 2014), (HUANG; YANG, 2008), (BEHNAMIAN, 2014), (BERKOUNE; MESGHOUNI, 2008), (LIU; LUH, 1996), (KUMAR et al., 2010), (AKHSHABI; AKHSHABI; KHALATBARI, 2011), (BARZEGAR; MOTAMENI; BOZORGI, 2012), (DOUSTHAGHI; MOGHADDAM; MAKUI, 2013), (LI; PAN, 2015), (SHARMA; JAIN, 2015), (KARABOGA et al., 2014)	
(JEONG et al., 1997), (DEFERSHA, 2011), (MORTEZAEI; ZULKIFLI, 2014), (PERES; LASSERRE, 1993), (LOW; HSU; HUANG, 2004), (KALIR; SARIN, 2000), (SAWIK, 2005), (AZZI et al., 2012), (RAMASESH et al., 2000), (SEN; BENLI, 1999), (BAKER, 1995), (WEEDA, 1990), (SEN; TOPALOGLU; BENLI, 1998), (HABCHI; LABRUNE, 1995), (CHENG; SARIN; SINGH, 2015), (ÇETINKAYA, 2006)	Trabalhos que se limitam à definição do modelo matemático sem propor métodos de resolução do FJSSP-LS. (16)
(VIJAYCHAKARAVARTHY; MARIMUTHU; SAIT, 2014), (DEFERSHA, 2015), (MARIMUTHU; PONNAMBALA, 2005), (XU et al., 2013), (LIU, 2011)	Trabalhos que não consideram operações no FJSSP-LS apresentando-o com apenas um estágio. (5)
(PAN et al., 2011_a), (PENG et al., 2014), (TSENG; LIAO, 2008), (KALIR; SARIN, 2001), (CHANG; JENG, 1995), (VENTURA; YOON, 2013), (YOON; VENTURA, 2002b), (PAN et al., 2011_b), (YALAOUI; CHU, 2003), (PAN; RUIZ, 2012), (CHAN; WONGB; CHAN, 2009), (BUSCHER; SHEN, 2011), (KALIR; SARIN, 2003), (CHIU; CHANG, 2005), (HONGYAN; LIANG; QUANKE, 2012), (CHAN; WONG; CHAN, 2004), (MARIMUTHU; PONNAMBALAM; JAWAHAR, 2008), (PETROVIC et al., 2008), (BUSCHER; SHEN, 2011), (ZHANG et al., 2005), (NASAB; SEYEDHOSEINI, 2013), (KALIR; SARIN, 2001), (CHAKARAVARTHY; MARIMUTHU; SAIT, 2013), (EDIS; ORNEK, 2009), (LIU, 2008), (TORABI; KARIMI; GHOMI, 2005), (KENYON; CANEL; NEUREUTHER, 2005), (OUENNICHE; BOCTOR, 2001), (HOSSEINABADI et al., 2015), (ASEFI et al., 2014), (PAN et al., 2011_c), (NASAB;	Trabalhos que têm como tema o FJSSP-LS com divisão de lotes de tamanhos iguais. (33)

MODARRES; SEYEDHOSEINICA, 2015), (CHAN; WONG; CHAN, 2008)	
(WONG; NGAN, 2013), (ROSSITA et al., 2016), (WONGA; CHAN; CHAN, 2009), (SARIN; KALIR; CHEN, 2008), (MARTIN, 2009), (BUSCHER; SHEN, 2009), (CHEN; STEINER, 1997), (CHEN; STEINER, 1998), (CHEN; STEINER, 1996), (DEFERSHA; CHEN, 2012), (YOON; VENTURA, 2002_c), (NEJATI et al., 2014), (CHEN; STEINER, 2003), (VICKSON, 1995), (ZHAO et al., 2009), (KIM; JEONG, 2009), (ÇETINKAYA; DUMAN, 2010), (HUANG, 2010), (GANAPATHY; MARIMUTHU; PONNAMBALAM, 2004)	Trabalhos que têm como tema o FJSSP-LS com divisão de lotes de tamanhos consistentes. (19)
(LIU, 2003), (CHAN; WONG; CHAN, 2005), (ROHANINEJAD; KHEIRKHAH; FATTAHI, 2015), (ZHAO et al., 2010)	Trabalhos que têm como tema o FJSSP-LS com divisão de lotes de tamanhos variáveis. (4)

Fonte: Elaborado pelo autor.

Também em (CAPES, 2016), foi feita uma pesquisa de maior espectro a fim de obter a quantidade total de trabalhos cujo tema era *Job Shop*, sem nenhuma restrição adicional de subdivisão de tema. Assim, através da Tabela 2 é possível obter a representatividade do tema divisão de lotes e mais especificamente da divisão de lotes de tamanho variável no contexto do tema *Job Shop*.

Tabela 2: Resumo dos trabalhos pesquisados

Quantidade de trabalhos encontrados sobre o tema *Job Shop*	2.596
Quantidade de trabalhos encontrados e analisados sobre o tema divisão de lotes	107
Quantidade de trabalhos encontrados e analisados sobre o tema divisão de lotes de tamanho variável	4

Fonte: Elaborado pelo autor.

O tema divisão de lotes representa apenas 4,12% do total de trabalhos relacionados a *Job Shop*. Além disso, o tema divisão de lotes de tamanho variável representa um percentual ainda menor, com apenas 0,15% em relação ao tema *Job Shop*. Mesmo dentre os autores que escreveram sobre divisão de lotes, somente 3,74% destes optou por explorar a divisão de lotes de tamanho variável.

A partir da pesquisa feita, foi possível verificar que o tema divisão de lotes de tamanho variável é pouco abordado na literatura existente.

2.6 Elementos considerados no modelo proposto

O JSSP é um problema extensamente discutido na literatura e com inúmeras propostas de solução (CALIS; BULKAN, 2015). Entretanto, a utilização destas soluções muitas vezes esbarra no fato de que foram concebidas com simplificações que inviabilizam sua implementação no mundo real (PENHA et al., 2014). Segundo GODINHO FILHO, BARCO e TAVARES NETO (2014), algoritmos clássicos são inadequados para o tratamento de ambientes SMF, pois respeitam um conjunto de premissas derivadas de problemas menos complexos. Assim, a implementação de um SMF requer o desenvolvimento de métodos específicos que considerem todas as premissas e restrições que descrevem o sistema. Do conjunto proposto de métodos de solução, técnicas de Inteligência Artificial (por exemplo, sistemas especialistas, Algoritmos Genéticos, Redes Neurais) provaram ser estratégias adequadas (GODINHO FILHO; BARCO; TAVARES NETO, 2014).

A seguir, serão enunciados os elementos indispensáveis a serem considerados na modelagem de um FJSSP-LS e cuja ausência diminui significativamente a aderência da solução a aplicações reais.

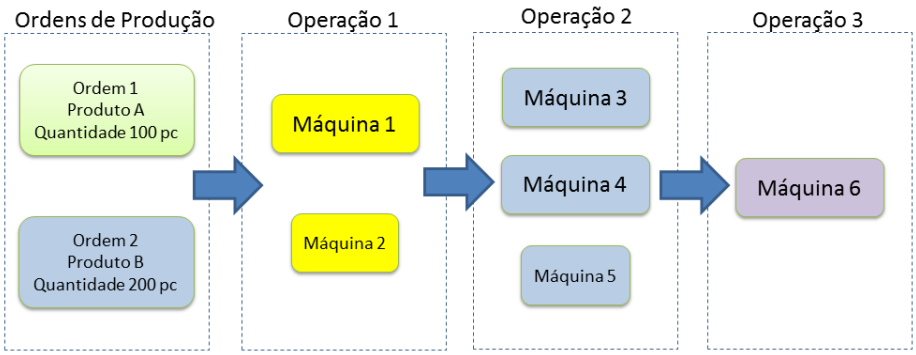
Uma importante contribuição deste trabalho é considerar todos esses elementos como parte integrante do modelo que é proposto, buscando assim soluções que reflitam as necessidades reais do SMF.

2.6.1 Divisão de lotes

No SMF, é muito comum a ocorrência de cenários onde existem várias máquinas capazes de executar as mesmas operações, trabalhando em paralelo com o objetivo de aumentar a produtividade. Assim, uma operação de uma ordem de produção pode ser encaminhada para ser executada em mais de uma máquina, reduzindo o tempo necessário para executar a operação em todas as peças. Para que isso seja possível, um lote de peças a processar deve ser dividido em lotes menores onde cada sublote é enviado a uma máquina para que seja processado.

Apesar da ideia simples de dividir o trabalho para finalizá-lo mais rápido, a implementação prática desse procedimento pode tornar-se bastante complicada, pois será necessário saber em quantas partições o lote será dividido e qual o tamanho de cada uma, levando-se em consideração a quantidade de máquinas que podem executar a operação, suas velocidades e a disponibilidade temporal de cada uma. A decisão ainda influenciará no escalonamento de outras operações de outras ordens de produção que utilizem as mesmas máquinas, podendo melhorar a entrega de uma operação e atrasar a produção de outras.

Figura 7: Exemplo de divisão de lotes no SMF



Fonte: Elaborado pelo autor.

A Figura 7 apresenta um exemplo de divisão de lotes no SMF, onde duas ordens de produção necessitam ser escalonadas e utilizarão as mesmas máquinas em suas três operações, cada uma com seus tempos de processamento. Pode-se observar que para a operação 1 existem duas máquinas que podem executá-la, sendo que o tamanho dos quadros, onde aparece a denominação da máquina, procura representar a velocidade em que a máquina trabalha, ou seja, quadros maiores indicam máquinas mais rápidas. Já a operação 2 tem 3 máquinas disponíveis para executá-la e a operação 3 apenas 1 máquina. Dessa forma, o problema de qual a divisão apropriada dos lotes se estabelece e é exatamente o problema que este trabalho se propõe a solucionar.

2.6.2 *Setup*

Setup é uma atividade de preparação da máquina que visa deixá-la em condições para produzir um determinado lote de peças, sendo o tempo necessário para essa atividade de preparação denominado como “tempo de *setup*”.

Dessa maneira, é possível identificar a condicionalidade da atividade de setup: sua existência depende se entre o lote finalizado e o novo lote há mudança de tipo de peça, pois se ambos os lotes são do mesmo produto, não há necessidade de *setup*. A condicionalidade se dá através de uma matriz de *setup* onde temos os produtos e operações que saem e os produtos e operações que entram. Assim, para cada relação, temos um tempo necessário para a atividade de *setup* e que deve ocorrer imediatamente antes do início de um lote.

As atividades de *setup*, em geral, se referem a programas a serem carregados nas máquinas, limpeza, troca de ferramental, entre outros.

2.6.3 Lote de transferência

No SMF, manter um fluxo contínuo de peças é uma necessidade para obter-se uma boa performance de produção, evitando acúmulo de peças dentro da fábrica. Para que isso seja possível, uma operação não pode esperar que todo o lote da operação predecessora seja finalizado para só então ser iniciada, pois muitas vezes existem lotes grandes em que o tempo de execução do lote pode levar horas ou até dias para ser finalizado. Assim, para que o fluxo de

produção tenha maior fluidez e para que menos estoques intermediários sejam criados entre operações, a utilização de lotes de transferência é indispensável no SMF.

O lote de transferência LT_{ij} é um parâmetro definido para cada operação de cada ordem de produção, tendo sua origem nas operações do roteiro de fabricação de cada produto. O lote de transferência, quando considerado na solução do FJSSP-LS, permite que uma operação seja iniciada logo após a operação predecessora tenha produzido quantidade igual ou superior ao número de unidades definidas pelo lote de transferência LT_{ij}. Com este procedimento adotado, as operações podem iniciar mais cedo (em relação às operações com ligação *end-start*, onde o início de uma depende do fim da outra), obtendo-se um menor tempo de execução final da ordem de produção.

2.6.4 Intervalos de indisponibilidade

No cenário de um SMF é praticamente impossível que o modelo seja aderente ao processo se ele não considerar que as máquinas têm períodos de indisponibilidade. Esses intervalos de indisponibilidade podem ser originários de fatores como: turnos de produção (onde o horário fora de turno é um intervalo de indisponibilidade), fins-de-semana, feriados, paradas para manutenção, entre outros.

Num primeiro momento, incluir os intervalos de indisponibilidade no modelo parece uma tarefa fácil, mas fazer com que a solução reflita corretamente essa restrição é bastante complicado, pois o escalonamento de um lote pode cruzar esses intervalos e assim será necessário expandir seu momento de finalização, através do incremento de um tempo igual a suas intersecções com os intervalos indisponíveis.

Outro efeito colateral negativo da consideração dos intervalos de indisponibilidade é a possibilidade de falta de fluxo de peças entre as operações, pois como as máquinas podem ter intervalos de indisponibilidade diferentes (devido a, por exemplo, regimes de trabalho diferentes) a taxa média de produção de uma operação pode ficar inferior à taxa da operação posterior e permitir, em determinado, momento a falta de fluxo, pois a operação posterior depende da disponibilidade das peças da operação anterior para poder ser executada. A própria composição dos lotes pode também criar esse tipo de problema. A Figura 8 mostra um exemplo do problema de falta de fluxo, onde na operação M2 no quarto dia (domingo) haverá falta de 40 peças para a execução da operação, mesmo sendo esta operação mais lenta do que a primeira operação M1, onde existe um período de indisponibilidade no sábado e no domingo.

Figura 8: Exemplo de falta de fluxo

Operação	Qui	Sex	Sab	Dom	Seg	Ter
M1	100	100	---	---	100	100
M2		80	80	80	80	80

Fonte: Elaborado pelo autor.

Na seção 5.1.1, é apresentado o método utilizado no modelo para evitar a ocorrência da falta de fluxo de peças entre as operações.

2.6.5 Máquinas alternativas

Uma das principais características do FJSSP é a possibilidade de executar uma mesma operação em mais de uma máquina, aumentando as alternativas de escalonamento das tarefas. Essa flexibilidade fica ainda mais importante quando se trabalha com a divisão de lotes, pois nesse caso as máquinas alternativas a uma operação poderão ser utilizadas simultaneamente para executá-la. Sendo assim, as máquinas alternativas são componentes importantes de um SMF que deseja otimizar seu escalonamento, através de um FJSSP-LS.

As máquinas alternativas podem ter características distintas no que tange ao tempo de processamento de um material de determinada operação, bem como operar com calendários de turnos distintos, não podendo ser considerados como um *pool* único de capacidade produtiva.

2.6.6 Recursos secundários

No SMF, é possível se trabalhar com cenários onde é importante não só escalonar de maneira correta e otimizada as máquinas, mas também outros recursos secundários como ferramentas, moldes, pessoas, entre outros. Nesse caso, cada recurso principal alocado poderá necessitar de um escalonamento do recurso secundário no mesmo período de tempo para que a operação possa ser executada.

É importante observar que os recursos secundários podem ter calendários de turnos diferentes do recurso principal, bem como planos de manutenção próprios que indisponibilizem o recurso por um determinado período.

2.7 Métodos de Resolução

O JSSP é um problema estabelecido há mais de dois séculos, tendo durante estes anos uma grande quantidade de trabalhos desenvolvidos para sua resolução, conforme visto no capítulo 2, utilizando as mais diversas técnicas e métodos. A seguir, são descritos três métodos de resolução do JSSP, a fim de possibilitar uma visão geral de como são as soluções existentes para esse importante problema no SMF.

2.7.1 Programação linear inteira utilizando Branch-and-Bound

A programação linear pode descrever uma imensa quantidade de problemas de otimização, sejam eles de maximização ou de minimização. Através do método Simplex, é possível resolver os problemas modelados com programação linear. Muitas vezes, é necessário trabalhar com variáveis inteiras e com variáveis binárias na modelagem do problema, levando à necessidade de utilizar a programação linear inteira, que pode ser resolvida através do método conhecido por *Branch-and-Bound*, onde o problema de programação inteira é subdividido em problemas relaxados de programação linear (não inteira), onde o método Simplex é aplicado para cada subproblema. A dificuldade de utilizar a programação linear inteira para resolver problemas de JSSP é que os métodos de resolução não são eficientes para um elevado número

de variáveis, e é exatamente isso que ocorre quando se modela um JSSP com maior número de máquinas ou de ordens de produção.

Em (BRUCKER; SCHLIE, 1990), é apresentado um modelo capaz de resolver o JSSP formulado com *n jobs* em *m* máquinas, onde cada *job* tem O_i operações que são processadas de maneira única em somente uma máquina. O método consegue valores ótimos para problemas com até 10 máquinas e 10 *jobs*, sendo que para maiores dimensões sua aplicabilidade é bastante limitada devido ao elevado tempo de processamento necessário para obter a solução.

2.7.2 Metaheurística de busca em trajetória utilizando Busca Tabu

A Busca Tabu (BT) é uma metaheurística baseada na busca em trajetória, onde a partir de uma solução são geradas outras soluções próximas (vizinhança), dentre as quais a melhor solução indica a trajetória a ser seguida pelo algoritmo. O processo segue ciclicamente, obtendo uma solução, gerando uma nova vizinhança e selecionando uma nova solução. Neste processo, um procedimento adotado é de criar uma lista de soluções proibidas que já foram visitadas e que impede que o algoritmo retorne a esses pontos. A essa lista é dado o nome de lista tabu, motivo pelo qual o método é chamado de Busca Tabu (GLOVER, 1989).

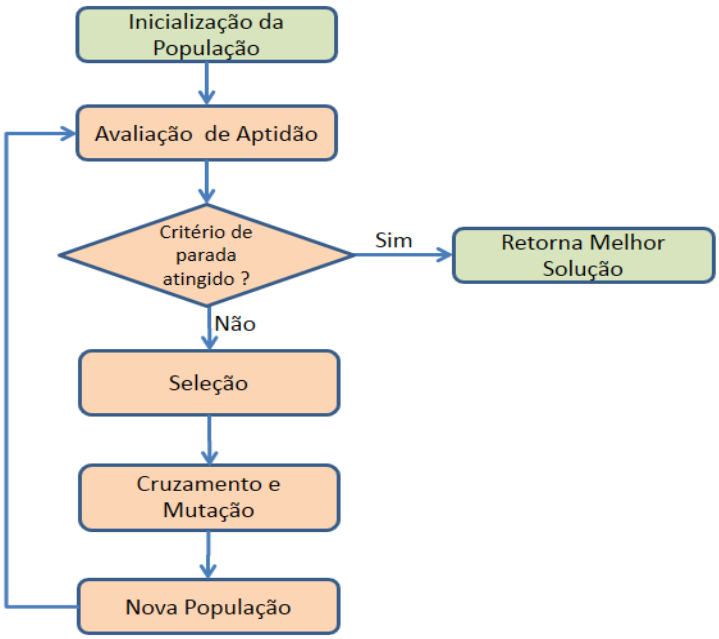
Existem muitos trabalhos na literatura sobre a resolução de JSSP utilizando BT, pois o método é capaz de trabalhar com problemas de maior dimensão, onde o número de máquinas e ordens de produção é grande, contendo dezenas, centenas, ou até mesmo milhares destes elementos. (BUSCHER; SHEN, 2009), (CHAMBERS; BARNES, 1996).

A BT busca soluções próximas ao ótimo, sendo que sua finalização é atingida a partir de um critério de parada previamente estabelecido.

2.7.3 Metaheurística evolutiva utilizando Algoritmo Genético

Um tipo de metaheurística extensamente estudada é a dos AGs, que tem seu funcionamento baseado na evolução natural dos seres vivos. Essa metaheurística traz bons resultados para problemas NP-Difíceis como o JSSP (GAREY, 1976). A Figura 9 apresenta uma visão geral de como o AG funciona, tendo sua ideia principal baseada na evolução das espécies na natureza, onde os indivíduos mais aptos têm maiores chances de sobreviver e transmitir suas características genéticas para as novas gerações (HOLLAND, 1975).

Figura 9: Algoritmos evolutivos



Fonte: Elaborado pelo autor.

O algoritmo inicia com uma população gerada aleatoriamente e através do processo de cruzamento e mutação são gerados novos indivíduos. Um processo de seleção, onde os mais aptos têm mais chances de sobreviver, se encarrega de reduzir a população novamente para o início de uma nova geração, e reiniciando o processo a partir desta nova população. Durante o processamento das gerações, os indivíduos são avaliados e aquele com melhor aptidão é armazenado como a melhor solução (GOLDBERG, 1989).

O AG, assim como a BT, busca soluções próximas ao ótimo, sendo que sua finalização é atingida a partir de um critério de parada previamente estabelecido.

3 MODELO PROPOSTO

O modelo a ser utilizado para obter soluções para o FJSSP-LS utilizará um algoritmo evolutivo, pois são uma classe de metaheurísticas extensamente estudadas e que trazem bons resultados para problemas NP-Difíceis como o JSSP (GAREY, 1976). A Figura 9 apresenta uma visão geral de como o algoritmo evolutivo funciona, tendo sua ideia principal baseada na evolução das espécies na natureza, onde os indivíduos mais aptos têm maiores chances de sobreviver e transmitir suas características genéticas para as novas gerações, além do aprendizado do indivíduo na população proporcionado pela busca local.

3.1 Modelo matemático do problema de escalonamento de lotes

O modelo matemático utilizado para a formulação do problema de escalonamento de lotes foi inspirado no modelo utilizado por (ZHAO et al., 2009), onde o JSSP foi resolvido utilizando um AG, considerando lotes de tamanho consistentes. Assim, várias adaptações foram necessárias para que o modelo considerasse todos os elementos do FJSSP-LS que se deseja abordar, utilizando lotes de tamanho variável.

$$Minimizar\ f = P_1\, TA\ +\ P_2\, TS\ +\ P_3\, TM\ , \tag{1}$$

onde:

$$TA = \sum_{i=1}^{N} \underset{\forall\, j,k}{\text{Max}}\{\, CWT_{ijk} - DDE_i\ , 0\,\}\quad ; \tag{2}$$

$$TS = \sum_{i=1}^{N}\sum_{j=1}^{Oi}\sum_{k=1}^{NB}\left(\, CPT_{ijk} - \ SPT_{ijk}\,\right)\ ; \tag{3}$$

$$TM = \underset{\forall\, i,j,k}{\text{Max}}\{\, CWT_{ijk}\}\quad ; \tag{4}$$

sujeito a:

$$\sum_{k=1}^{NB}\left(\, QL_{ijk}\,\right) = \ Q_i\ \ ,\ \ \forall\, i\,, j\quad ; \tag{5}$$

$$SWT_{ijk} >= \underset{\forall\, k}{\text{Min}}\{\, SWT_{ij-1k}\} \ +\ LT_{ij-1} *\ \ TMP_{ij-1l}\quad \forall\, i\,, j, k\ \ e\ \ j > 1\quad ; \tag{6}$$

$$CWT_{ijk}\ =\ SWT_{ijk} + TMP_{ijl} * QL_{ijk} + \left(\, SWT_{ijk}\,, CWT_{ijk}\right) \cap \left(\, LL_{lq}, LU_{lq}\right)\ \ \forall\ \ i\,, j\,, k\,, q\quad ; \tag{7}$$

$$CPT_{ijk} = SWT_{ijk} \quad \forall\, i\,, j\,, k \quad ; \tag{8}$$

$$SPT_{ijk} = CPT_{ijk} - TMS_{ijl} - \left(SPT_{ijk}\,, CPT_{ijk}\right) \cap \left(LL_{lq}, LU_{lq}\right) \quad \forall\, i\,, j\,, k\,, q \quad ; \tag{9}$$

$$SE\ OM_{ijk} = OM_{i'j'k'} \begin{cases} \left(SWT_{ijk}\,, CWT_{ijk}\right) \cap \left(SWT_{i'j'k'}\,, CWT_{i'j'k'}\right) = \emptyset \quad ; \\ \left(SPT_{ijk}\,, CPT_{ijk}\right) \cap \left(SPT_{i'j'k'}\,, CPT_{i'j'k'}\right) = \emptyset \quad ; \\ \left(SWT_{ijk}\,, CWT_{ijk}\right) \cap \left(SPT_{i'j'k'}\,, CPT_{i'j'k'}\right) = \emptyset \quad ; \\ \forall\ i\,, j\,, k\,, i', j', k'\ com\ (i,j,k) \neq (i', j', k') \quad . \end{cases} \tag{10}$$

Índices, parâmetros e variáveis do modelo:

i	Índice da ordem, com $i \in \{ 1,...,N \}$
j	Índice da operação, com $j \in \{ 1,...,O_i \}$
k	Índice do lote, com $k \in \{ 1,...,NB \}$
l	Índice da máquina, com $l \in \{ 1,...,M \}$
N	Número de ordens de produção
O_i	Número de operações da ordem *i*
M	Número de máquinas
Q_i	Quantidade de peças da ordem *i*
NB	Número máximo de lotes que uma operação pode ser dividida
DDE_i	Data desejada de entrega da ordem *i*
LL_{lq}	Início do *q*-ésimo intervalo de indisponibilidade da *l*-ésima máquina
LU_{lq}	Término do *q*-ésimo intervalo de indisponibilidade da *l*-ésima máquina
SWT_{ijk}	Início do *k*-ésimo lote da *j*-ésima operação da *i*-ésima ordem
CWT_{ijk}	Término do *k*-ésimo lote da *j*-ésima operação da *i*-ésima ordem
SPT_{ijk}	Início do *setup* do *k*-ésimo lote da *j*-ésima operação da *i*-ésima ordem
CPT_{ijk}	Término do *setup* do *k*-ésimo lote da *j*-ésima operação da *i*-ésima ordem
LT_{ij}	Tamanho do lote de transferência da *j*-ésima operação da *i*-ésima ordem
QL_{ijk}	Tamanho do *k*-ésimo lote da *j*-ésima operação da *i*-ésima ordem
TMP_{ijl}	Tempo de processamento da *j*-ésima operação da *i*-ésima ordem na máquina *l*
TMS_{ijl}	Tempo de *setup* da *j*-ésima operação da *i*-ésima ordem na máquina *l*
OM_{ijk}	Máquina usada pelo do *k*-ésimo lote da *j*-ésima operação da *i*-ésima ordem
P_1	Peso do índice de atraso de cada ordem
P_2	Peso do índice de tempo de setup
P_3	Peso do índice do *makespan*
TA	Tempo total de atraso
TS	Tempo total de *setup*
TM	Tempo total de produção (*makespan*)

Descrição do modelo:

(1) Cálculo da função objetivo, onde deseja-se minimizar a soma do tempo total de atraso de cada ordem de produção (*TA*), do tempo total de *setup* (*TS*) e do tempo total de produção (*TM*), sendo que cada parcela está sujeita a um multiplicador (P_1, P_2, P_3), que indica qual o peso que cada tempo terá na função objetivo;

(2) Cálculo do tempo total de atraso (*TA*), que é o somatório dos tempos de atraso individuais de cada ordem de produção. O tempo de atraso é calculado pela diferença entre data de conclusão da última operação da ordem de produção e de sua data desejada de entrega. Este valor só é considerado no cálculo caso tenha resultado maior que zero, indicando que a ordem de produção está escalonada com atraso;

(3) Cálculo do tempo total de setup (*TS*), que é o somatório de todos os tempos de *setup* escalonados. O tempo de setup é calculado pela diferença entre o fim de uma atividade de *setup* e seu início;

(4) Cálculo do tempo total de produção (*TM*), que é o tempo compreendido entre o início e o fim do escalonamento, onde o fim é determinado pela última operação da última ordem de produção;

(5) Restrição que indica que a soma da quantidade de todos os lotes de uma operação em cada ordem deve ser igual à quantidade da ordem de produção;

(6) Restrição que indica que um lote de uma operação poderá iniciar somente após a operação predecessora tiver produzido uma quantidade igual ao seu lote de transferência.

(7) Cálculo da data de término de um lote, que tem por base a data de início do lote adicionado do tempo de processamento unitário multiplicado pela quantidade de peças do lote, adicionando ao cálculo as intersecções do período de produção do lote com intervalos de indisponibilidade, pois nesses intervalos não há processamento de peças;

(8) Cálculo da data de término de uma tarefa de *setup*, que será igual ao início de produção do lote;

(9) Cálculo da data de início de uma tarefa de *setup*, que tem por base a data de término do *setup* subtraído do tempo de *setup* necessário na máquina/operação. São adicionadas ao cálculo as intersecções do período de *setup* com intervalos de indisponibilidade, pois nesses intervalos não há execução de *setup*;

(10) Restrições que indicam que não pode haver sobreposição, em uma mesma máquina, de períodos de processamento ou períodos de *setup*. Caso isso seja detectado, o período é deslocado até atender a restrição.

Regras adicionais:

- Ao escalonar um lote, é necessário que seja avaliada a taxa de produção dos lotes de operações predecessoras e todos os lotes já escalonados para a operação corrente, a fim de evitar que ocorra falta de abastecimento de peças no decorrer da execução de algum lote (vide seção 5.1.1);

- O índice *l* (referente à máquina *l*) é obtido para cada lote através da variável OM_{ijk} que armazena a máquina selecionada para produzir o lote.

3.2 Função objetivo

Para cada solução será necessário extrair o índice que representa sua qualidade ou aptidão. Isto é feito através da função objetivo, que é composta por três variáveis de decisão, cada uma com seu peso.

- **Tempo total de atraso**: é a soma do atraso individual de cada ordem, que por sua vez é a diferença entre tempo final de conclusão do último lote da última operação da ordem e a data desejada de entrega da ordem. Caso esse indicador seja negativo, indicando que a ordem foi entregue antes da data desejada, será considerado como valor zero na função objetivo;
- **Tempo total de *setup***: é a soma de todos os tempos das operações de *setup* do escalonamento;
- ***Makespan***: é o tempo necessário para conclusão de todas as ordens, ou seja, o intervalo de tempo entre o início da primeira ordem até a conclusão da última operação da última ordem processada (LUSTOSA, 2008).

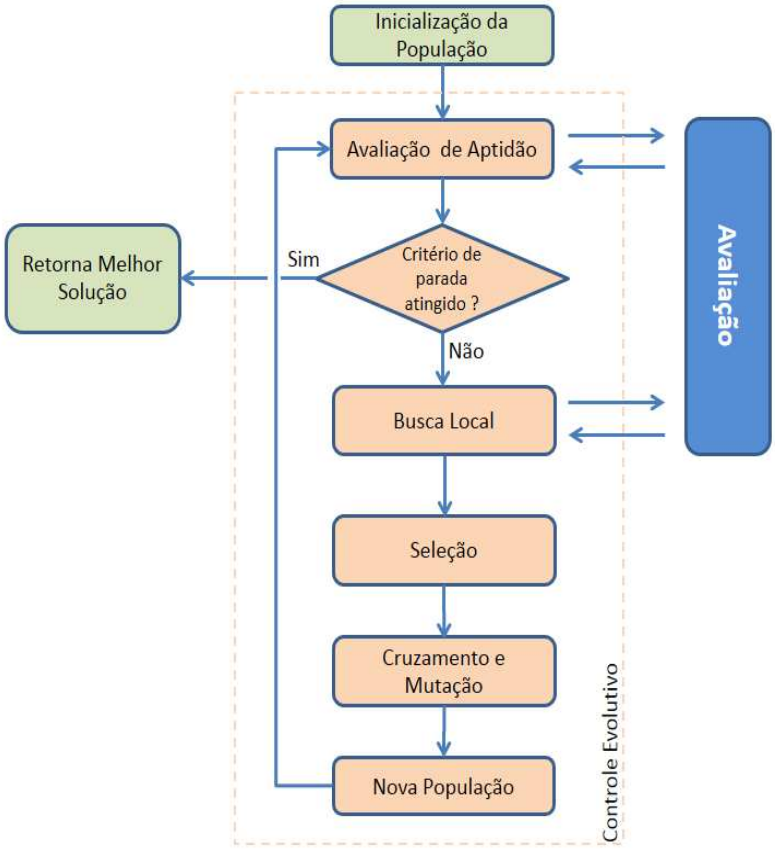
A cada um desses índices é aplicado um peso e a soma de todas as parcelas resulta no valor da função objetivo.

3.3 Estrutura computacional

A implementação do modelo computacional proposto divide-se basicamente em duas partes distintas:

- **Rotina de controle da parte evolutiva**: responsável pelo processamento das gerações, manutenção da população, cruzamentos entre indivíduos, mutações dos cromossomos, busca local, aplicação da roleta de seleção para selecionar a nova geração e verificação do critério de parada;
- **Rotina de avaliação dos indivíduos**: responsável pela transformação do par de cromossomos numa lista de sequenciamento contendo o tempo inicial e final de cada lote de cada operação de cada ordem de produção. Esta é uma rotina complexa devido às características implementadas, como: lotes variáveis, recursos secundários, intervalos de indisponibilidade e lotes de transferência. Várias técnicas de programação são necessárias, como programação para frente e programação para trás *(forward and backward scheduling*).

Figura 10: Fluxo do Algoritmo Memético



Fonte: Elaborado pelo autor.

A Figura 10 apresenta o fluxo do AM, onde a rotina de avaliação aparece em destaque em azul à direita e as rotinas de controle evolutivo ao centro. O bloco do controle evolutivo executa os procedimentos de cruzamento, mutação, busca local e seleção nos indivíduos da população, até que o critério de parada seja atingido. Para cada indivíduo, em cada geração, é acionada a rotina de avaliação para que se obtenha o valor da função objetivo, que representa a aptidão do indivíduo. A busca local também utiliza a rotina de avaliação para identificar se há melhora no aprendizado do indivíduo.

3.4 Representação da solução

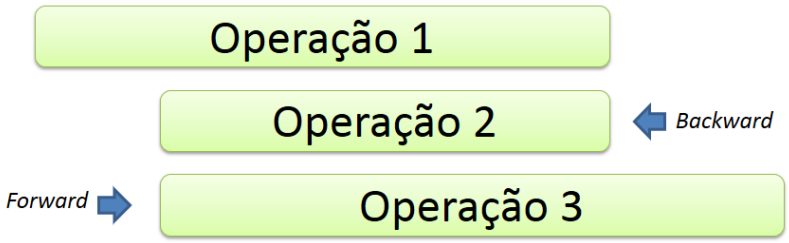
Para a representação da solução, o problema reside em converter o par de cromossomos, que são detalhados na seção 3.5.1, no quadro de sequenciamento (gráfico de GANTT (1913) onde são alocados os recursos segundo as regras de sequenciamento), pois é necessário levar em consideração fatores como:

- **Utilização dos recursos**: não poderá haver sobreposição na utilização dos recursos, isto é, cada máquina só pode processar um trabalho de cada vez;
- **Tempo de *setup***: se uma máquina mudar de produto ou de ferramenta, deverá ser alocado o tempo de *setup* antes do início da nova tarefa;

- **Intervalos de indisponibilidade**: quando um lote interseccionar um ou mais intervalos de indisponibilidade, ele deve considerar que não há produção no intervalo de indisponibilidade, expandindo o tempo alocado até que o lote tenha tempo disponível suficiente para ser produzido.
- **Recursos secundários (ferramentas)**: caso uma operação indique a necessidade da utilização de um recurso secundário (como uma ferramenta), a alocação de uma máquina deverá estar ligada à alocação deste recurso. Ou seja, deve haver tempo disponível no mesmo intervalo para o recurso principal e para o secundário (ao mesmo tempo).

Uma grande diferença entre o método de escalonamento utilizado para montar o quadro de sequenciamento e o que tradicionalmente se encontra nos trabalhos de JSSP, é que para cada operação há um parâmetro denominado quantidade de transferência (ou lote de transferência), onde após produzir essa determinada quantidade na operação, a operação seguinte já pode iniciar seu processamento, não necessitando esperar que todas as peças da operação anterior estejam finalizadas (WAGNER; RAGATZ, 1994). Isso é um fator que reflete a realidade do que acontece em um SMF com produção discreta, gerando assim um escalonamento mais eficiente e mais próximo da realidade.

Figura 11: Exemplo de programação para frente e para trás



Fonte: Elaborado pelo autor.

A Figura 11 apresenta um exemplo de programação para frente e de programação para trás, onde são apresentadas três operações em que a segunda operação, por ter um tempo de processamento mais curto que a primeira operação, é programada para trás, ou seja, tendo seu fim baseado no fim da primeira operação e o início calculado através da subtração do tempo de processamento. Já a terceira operação, por ter um tempo de processamento maior que a segunda operação, é programada para frente, ou seja, tendo seu início baseado no início da segunda operação e o final calculado adicionando o tempo de processamento. A programação para frente e a programação para trás estabelecem relações entre as operações que são conhecidas como relação início-início e relação fim-fim.

A geração da lista de sequenciamento é feita com base no cromossomo onde está a sequência em que as ordens de produção e operações devem ser executadas. Cada gene do cromossomo é processado a fim de verificar em qual máquina a referida operação pode ser processada. Com a utilização de máquinas alternativas, tem-se várias possibilidades de alocação, onde a máquina escolhida será aquela que proporcionar o término mais cedo da operação (KIS, 2014). Esta rotina de avaliação consome 95% do tempo total de processamento do AM, sendo um ponto crítico na execução do programa.

Após alocar todas as operações nas devidas máquinas, o programa localiza o lote temporalmente mais avançado da última operação de cada ordem de produção a fim de obter o

momento final global do escalonamento, calculando assim o *makespan*, que é o tempo total de produção.

3.5 Metaheurística evolutiva utilizando Algoritmo Memético

Na classe das metaheurísticas evolutivas, os AGs sempre tiveram posição de destaque, pois são extensamente estudados e trazem excelentes resultados na solução de problemas NP-Difíceis (GODINHO FILHO; BARCO; TAVARES NETO, 2014). Dentre suas características mais importantes pode-se destacar a capacidade exploratória do espaço de busca e a intensificação da busca em determinado local do espaço, sendo o caráter exploratório mais predominante do que a intensificação, levando o algoritmo a ter uma boa performance para descobrir locais promissores no espaço de busca e uma dificuldade de conseguir uma boa convergência para a obtenção do refinamento de melhores resultados nas buscas locais (PATALIA; KULKARNI, 2010).

A proposta do AM é prover uma melhora na busca local de soluções em relação ao AG, mantendo também o já eficaz caráter exploratório do algoritmo. A ideia de memes (DAWKINS, 1976) parte do princípio de que um indivíduo pode obter uma evolução cultural através do autoconhecimento e do contato com outros indivíduos, obtendo um progresso que não advém somente de suas características hereditárias. Essa ideia de utilização de memes para o aprimoramento da busca local foi formalizada por (RADCLIFFE; SURRY, 1994), onde apresentaram um modelo de utilização para o AM, termo este que já havia sido introduzido por (MOSCATO; NORMAN, 1992).

Figura 12: Algoritmo Memético

The Memetic Algorithm

```
1  begin
2      foreach S in Population do S ← LocalSearch(Init());
3      while not terminated do
4          Offspring ← ;
5          for i ← 0 to crossovers do
6              A ← Select(Population);
7              B ← Select(Population);
8              C ← LocalSearch(Recombine(A, B));
9              Offspring ← OffSpring + C;
10         endfor
11         for i ← 0 to mutations do
12             A ← Select(Population);
13             C ← LocalSearch(Mutate(A));
14             Offspring ← OffSpring + C;
15         endfor
16         Population ← Select(Population, Offspring);
17     endw
18 end
```

Fonte: NERI, 2012.

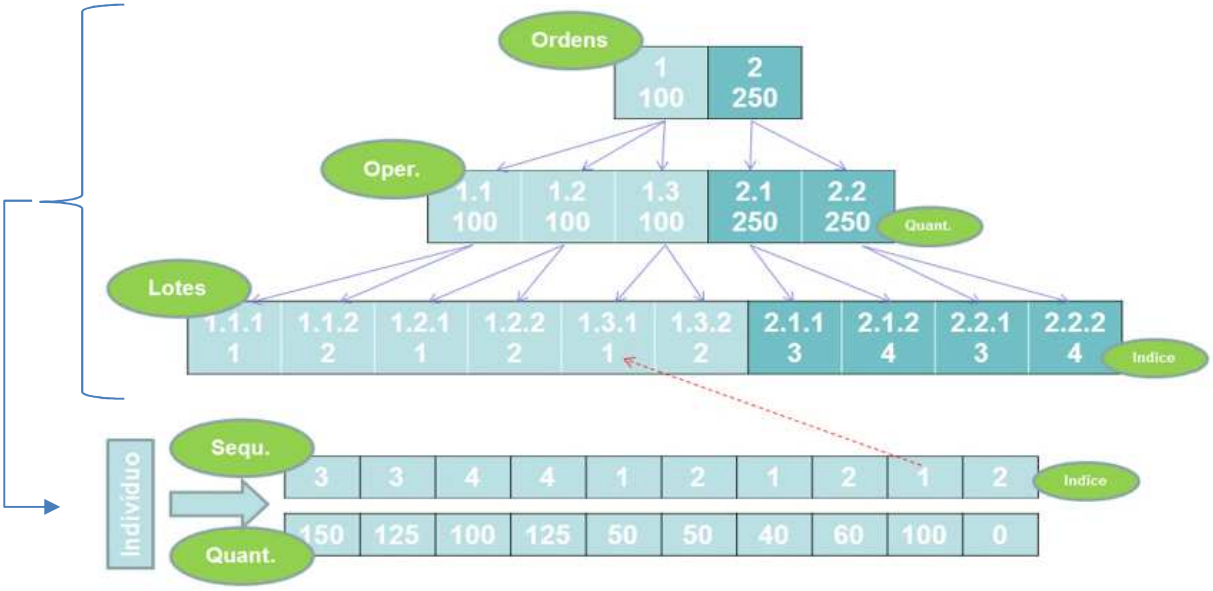
A Figura 12 apresenta um exemplo de AM, onde a inicialização da população é influenciada por uma busca local, de modo que o algoritmo já inicie com uma população mais capacitada (soluções melhores). O processo de evolução está baseado no processamento de gerações, onde a população é afetada por cruzamentos e mutações, além de sofrer influência da busca local, onde o auto aprendizado e o aprendizado entre os indivíduos contribui para o aprimoramento da população. Ao final de cada geração, é aplicado um processo de seleção, onde os indivíduos mais aptos têm maior probabilidade de serem selecionados para a próxima geração. Após, uma nova geração é iniciada e o processo evolutivo é repetido até que o critério de parada desejado seja atingido.

3.5.1 Codificação dos cromossomos

No modelo proposto, um indivíduo será constituído por um par de cromossomos, onde um cromossomo possui as informações sobre o sequenciamento dos lotes das operações das ordens de produção e o outro cromossomo contém as informações relativas ao tamanho de cada lote que compõem as operações de cada ordem de produção. Assim, um indivíduo com seus dois cromossomos representa uma solução viável para o problema do FJSSP-LS.

No primeiro cromossomo está representada a sequência em que os lotes devem ser sequenciados. Cada gene do cromossomo indica um lote onde seu valor é um índice referente à ordem, operação e lote a que pertence. A cada dupla única, composta por uma ordem e um lote, é dado um índice que a representa. Assim, o índice se repete para operações diferentes em uma mesma ordem e mesmo lote. A operação é representada pela enésima ocorrência do índice. Dessa forma, por exemplo, quando um índice *n* ocorrer pela 3ª vez, ele estará indicando a 3ª operação de uma determinada ordem e lote indicada pelo índice *n*.

Figura 13: Mapeamento dos cromossomos do modelo



Fonte: Elaborado pelo autor.

No segundo cromossomo, está representada a quantidade de cada lote de cada operação de cada ordem. Os genes estão dispostos em ordem crescente de ordem de produção, operação e lote.

A Figura 13 apresenta um exemplo da estrutura que as ordens de produção, operações e lotes ocorrem no cenário real e como um indivíduo é representado pelo par de cromossomos. Na figura, a seta pontilhada indica a relação do 9º gene, contendo o índice “1”, com o primeiro lote da terceira operação da primeira ordem (1.3.1). É possível observar que este gene é a terceira ocorrência do índice “1” no cromossomo, logo estará relacionado a terceira ocorrência do índice “1” na linha onde são apresentados os lotes.

3.5.2 Soluções iniciais

O início do processo evolutivo dos AMs é, diferentemente da natureza, um processo criacionista. Isso porque os indivíduos da primeira população são criados numa espécie de abiogênese em que, a partir dos possíveis valores para os genes, são selecionados aleatoriamente valores para compor um indivíduo. Porém não há a obrigatoriedade de que todos os indivíduos sejam criados apenas pela aleatoriedade, podendo-se criar alguns com características que, supostamente, lhes proporcionem maior aptidão e possibilidade de sobrevivência. AMs tipicamente usam soluções iniciais boas como ponto de partida (NERI, 2012).

Figura 14: Algoritmo de inicialização da população

Injecting high-quality solutions in the initial population.

```
1  function Initialize(in par: Parameters, in P: Problem): Bag{Solution};
2  begin
3      pop ← ∅;
4      for j ← 1 to par.popsize do
5          i ← RandomSolution(P);
6          i ← LocalSearch (i, par, P);
7          pop ← pop ∪ {i};
8      endfor
9      return pop;
10 end
```

Fonte: NERI, 2012.

A Figura 14 apresenta um exemplo de rotina de inicialização da população, onde cada indivíduo é criado numa combinação de aleatoriedade e aprimoramento pela busca local.

Na solução inicial do algoritmo implementado, um dos cromossomos é criado para refletir um sequenciamento que representa as tarefas em ordem crescente de data desejada de entrega, para que essa informação seja avaliada e compartilhada através da combinação com outros indivíduos.

3.5.3 Operadores de cruzamento

Um dos objetivos das metaheurísticas evolutivas é criar novas gerações com indivíduos, de preferência, cada vez mais aptos e que assim tenham mais chances de que seus descendentes,

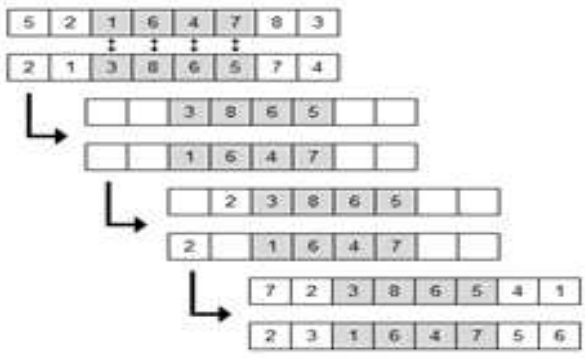
e ele próprio, integrem a nova geração. É neste contexto que os operadores de cruzamento estão inseridos: eles são necessários para guiar como será feito o procedimento de geração de novos indivíduos. Assim, um operador de cruzamento age sobre dois (ou mais) indivíduos da população e gera um (ou mais) novo indivíduo com características herdadas de seus pais.

Existem muitos tipos de operadores de cruzamento, criados à medida que as técnicas dos AGs foram evoluindo para obterem melhores resultados em determinados tipos de problemas. Os principais operadores de cruzamento são:

- **Cruzamento simples**: este operador executa a operação de cruzamento da maneira mais simples possível, onde os novos cromossomos gerados têm uma parte contínua (de posição e tamanho aleatório) que é trocada entre eles. Essa técnica somente pode ser utilizada para modelos de cromossomos cujo cruzamento simples não implique na geração de filhos degenerados (com informações duplicadas, por exemplo), motivo pelo qual não poderá ser utilizado no modelo implementado;

- ***PMX Crossover (Partial Mapped Crossover)***: o operador de cruzamento parcialmente mapeado (GOLDBER; LINGLE, 1985) gera dois novos cromossomos com uma parte contínua (de posição e tamanho aleatório) que é trocada entre eles. No restante dos genes é feito um mapeamento entre os cromossomos pais a fim de verificar a existência daquela característica no cromossomo filho, evitando duplicações de valores nos genes;

Figura 15: Operador de cruzamento PMX



Fonte: Elaborado pelo autor.

Na implementação do algoritmo, é utilizado o operador PMX, exemplificado na Figura 15, onde dois cromossomos são submetidos ao operador PMX e tem seus genes trocados da posição 3 até a posição 6. Nas outras posições é feito um mapeamento para evitar que valores já utilizados (devido à troca) sejam repetidos.

3.5.4 Operadores de mutação

A evolução dos indivíduos através das gerações, proporcionada pela reprodução com cruzamento, tende a manter somente as características que já existem na população, ficando prezo em mínimos locais. Assim, novas habilidades que ajudem na solução do problema não

poderão surgir. Para que essa estagnação não ocorra, os algoritmos evolutivos utilizam, novamente, um processo semelhante ao existente na natureza que é a mutação, cuja finalidade é proporcionar novas características aos novos indivíduos gerados, sejam elas boas ou ruins, diversificando as aptidões para além daquelas previamente existentes na população. Em (ABDOUN; ABOUCHABAKA; TAJANI, 2012) são descritos vários operadores de mutação, entre eles:

- **Mutação aleatória**: este operador seleciona aleatoriamente um gene (ou um conjunto de genes) do cromossomo e altera, também aleatoriamente, seu valor, respeitando o intervalo de valores que o gene pode assumir. Este operador de mutação não é compatível com o modelo implementado, pois a mutação aleatória de um gene implicaria em indivíduos degenerados (com informações duplicadas, por exemplo).

- **Mutação por troca**: este operador (também conhecido como operador de *swap*) seleciona aleatoriamente duas partes do cromossomo e efetua a troca de um bloco de genes pelo outro. Normalmente, os blocos têm tamanhos pequenos, com 1 ou 2 genes.

- **Mutação por inserção**: este operador seleciona aleatoriamente um gene (ou um conjunto de genes) do cromossomo e move-o para outra posição, também aleatória, do cromossomo;

Figura 16: Operadores de mutação por troca e por inserção

Fonte: Elaborado pelo autor.

A Figura 16 apresenta (à esquerda) um exemplo do operador de mutação por troca, onde foram selecionados os genes 2 e 7 e executada a troca de posição entre eles, resultando no cromossomo modificado. Na figura também é apresentado (à direita) um exemplo do operador de mutação por inserção, onde foi selecionado o gene 2 para ser movido para a quinta posição no cromossomo. Na implementação do algoritmo, são utilizados os operadores de mutação por troca e por inserção, com tamanho igual a 1 gene em ambos.

3.5.5 Métodos de seleção

Como as metaheurísticas evolutivas baseiam-se no processo de evolução natural, é necessário que seja definido qual método será utilizado para determinar quais indivíduos farão parte da geração seguinte. Esse é o objetivo do método de seleção: selecionar indivíduos baseando-se em determinados critérios que envolvem aptidão, aleatoriedade e probabilidade. Existem inúmeros métodos de seleção, entre os quais os mais utilizados são (REEVES, 2003):

- **Roleta de seleção**: semelhante a uma roleta de um cassino, que está dividida em seções de iguais tamanhos que representam os números, a roleta de seleção segue a mesma ideia, porém a divisão das seções não é de igual tamanho, mas sim

proporcionais à aptidão de cada indivíduo. Assim, cada indivíduo tem uma probabilidade de seleção igual a sua aptidão dividida pela soma das aptidões de todos os indivíduos. O objetivo é unir a aleatoriedade da escolha com a probabilidade devido à aptidão do indivíduo;

- **Seleção por classificação**: semelhante ao que é feito no método da roleta de seleção, a seleção por classificação segue a mesma ideia, porém a divisão das seções não é proporcional à aptidão de cada indivíduo, mas sim a ordem inversa da ordenação dos indivíduos pela sua aptidão. Ou seja, o indivíduo mais apto tem índice *N*, o segundo *N-1* e assim sucessivamente até o último indivíduo cujo índice será *1*. Assim, cada indivíduo tem uma probabilidade igual a sua ordem inversa dividida pelo somatório das ordens (somatório de *N*). O objetivo é unir a aleatoriedade da escolha com uma probabilidade mais uniforme;

- **Seleção por torneio**: este método consiste em sortear aleatoriamente os indivíduos para compor um grupo e selecionar aquele com maior aptidão dentre os integrantes do grupo. O número de componentes do grupo é um parâmetro informado e deve-se executar o procedimento do torneio (sorteio, agrupamento e seleção) tantas vezes quantos forem os indivíduos desejados na nova população.

Dentre os métodos de seleção apresentados, a roleta de seleção é o método utilizado em nosso AM.

3.5.6 Critérios de parada

Uma característica das metaheurísticas é que, na maioria dos casos, não há como saber se um máximo ou mínimo global foi atingido. Dessa forma, o algoritmo continuará buscando uma solução melhor indefinidamente, o que certamente não é desejado. Para evitar esse problema, o algoritmo estará sujeito a uma verificação periódica, onde se um determinado critério é atingido, a busca por melhores soluções é encerrada e o algoritmo termina. Esse critério de verificação é chamado de critério de parada e pode se apresentar de várias formas, algumas delas descritas a seguir:

- **Limite de número de gerações**: indica ao algoritmo que o número de gerações a ser executado na busca da melhor solução é limitado a um determinado valor informado;

- **Limite de tempo de execução**: indica ao algoritmo que o tempo de execução utilizado na busca da melhor solução é limitado a um determinado valor informado;

- **Limite de índice desejado da função objetivo:** indica ao algoritmo que finalize a busca pela melhor solução ao obter da função objetivo o limite igual ou melhor que um determinado valor informado. Esse critério de parada pode fazer com que o algoritmo não seja finalizado nunca, caso não se atinja o limite estabelecido;

- **Limite de tempo sem melhora da solução**: indica ao algoritmo que o tempo de execução após encontrar uma melhora na solução é limitado a um determinado

valor informado. A contagem desse tempo sem melhora reinicia cada vez que o algoritmo encontra uma nova solução melhorada;

- **Limite de gerações sem melhora da solução**: indica ao algoritmo que o número de gerações executadas após encontrar uma melhora na solução é limitado a um determinado valor informado. A contagem desse número de gerações sem melhora reinicia cada vez que o algoritmo encontra uma nova solução melhorada.

O critério de parada pode ainda ser adotado como uma composição de dois outros critérios, como, por exemplo, limitar o processamento em 1 hora e limitar o tempo sem melhora da solução em 10 minutos. No SMF é muito comum a adoção do critério de parada por tempo de execução, pois assim sabemos quando o resultado estará disponível. Já em trabalhos acadêmicos, o critério de parada por número de gerações é o mais adotado, devido à necessidade de comparação entre trabalhos relacionados.

3.5.7 Elitismo

Considerando que as metaheurísticas tem um caráter fortemente probabilístico, muitas vezes deseja-se que, assim como na vida real, certos indivíduos sejam selecionados independentemente do método de seleção utilizado, configurando-se uma elite privilegiada, motivo pela qual este processo é chamado de elitismo.

Normalmente, o elitismo seleciona os *Ne* mais aptos para comporem a próxima geração, onde *Ne* é um parâmetro informado e pode variar de 0 ao número de indivíduos da população. Caso *Ne* seja igual a zero, não há elitismo. No modelo apresentado, *Ne* será igual a 1, ou seja, manter-se-á apenas o melhor indivíduo para a próxima geração.

3.5.8 Busca local: o aprendizado como evolução

A principal característica que diferencia um AM dos outros algoritmos evolutivos é a presença de uma etapa onde a busca local é empregada com o propósito de evoluir os indivíduos da população, através de um processo de aprendizado entre os indivíduos.

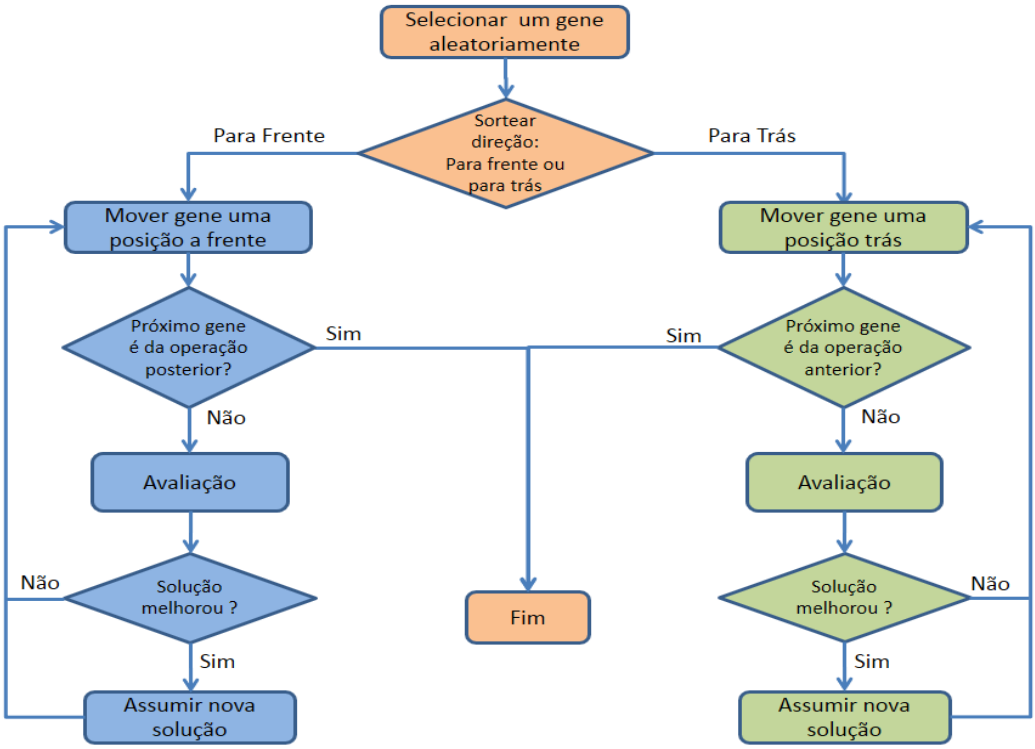
O método de busca local baseado em inserção (ZHAO et al., 2010), onde o indivíduo reconhece, a partir de suas características, a necessidade de realocação de seus genes, será utilizado em nosso AM para executar a busca local.

O princípio básico desde método de busca local é escolher um gene aleatoriamente no cromossomo de sequenciamento e movê-lo para frente (ou para trás) até encontrar um gene que representa a operação posterior (ou anterior). O gene é movido uma posição de cada vez no cromossomo e para cada movimento a função de avaliação é executada. Caso o resultado da função objetivo seja melhor que a original, a solução é aceita e o cromossomo assume esta configuração.

A Figura 17 apresenta o fluxo de funcionamento da busca local utilizando o método de inserção. Nela é possível observar dois blocos de processamento distintos: o bloco à direita,

responsável pelo movimento do gene para trás e o bloco à esquerda, responsável pelo movimento do gene para frente.

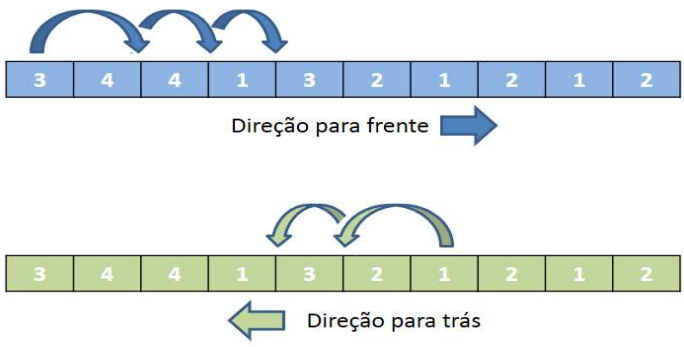
Figura 17: Fluxo de funcionamento da busca local



Fonte: Elaborado pelo autor.

Na rotina de busca local por inserção, se o gene selecionado for um gene da primeira operação de uma ordem de produção, então somente a opção de movimento para frente estará disponível, pois não existe operação anterior à primeira operação. De forma análoga, se o gene selecionado for um gene da última operação de uma ordem de produção, então somente a opção de movimento para trás estará disponível, pois não existe operação posterior a última operação. A Figura 18 apresenta um exemplo do funcionamento da busca local.

Figura 18: Exemplo da busca local



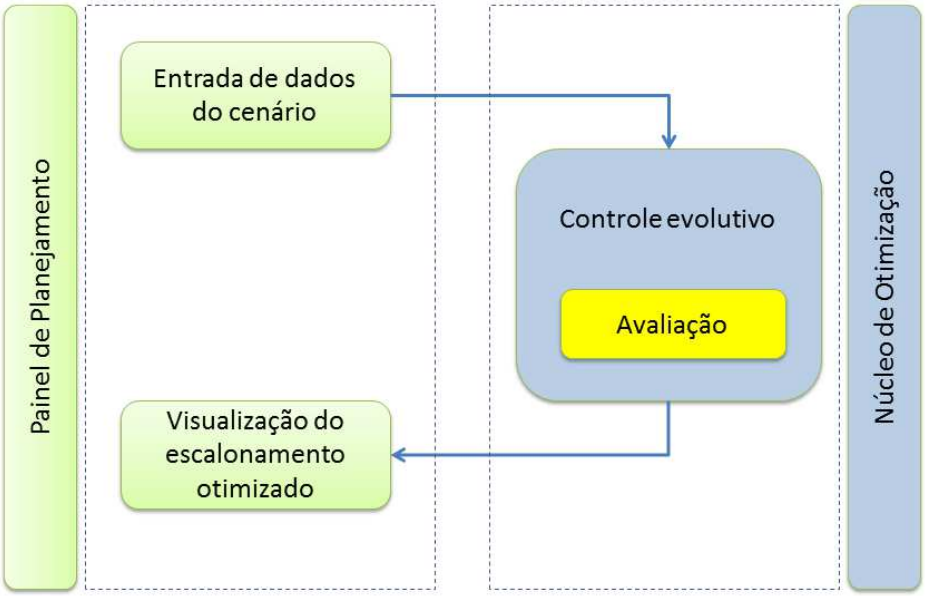
Fonte: Elaborado pelo autor.

4 IMPLEMENTAÇÃO

A implementação do modelo está dividida em dois módulos: a primeira parte é o núcleo de otimização e a segunda parte é o painel de planejamento.

Os dois módulos têm funções bem distintas, pois o núcleo de otimização é responsável pelo processamento e evolução das gerações, gerando a solução do FJSSP-LS. Já o painel de planejamento tem como função a organização do cenário de avaliação com os elementos de planejamento e a posterior visualização da solução gerada pelo núcleo de processamento.

Figura 19: Visão geral da implementação



Fonte: Elaborado pelo autor.

A Figura 19 mostra como os dois módulos interagem na criação, otimização e visualização do cenário.

4.1 Núcleo de otimização

O núcleo de otimização é o módulo onde está implementado o AM. Foi desenvolvido em linguagem C para obter-se a melhor performance possível, bem como a portabilidade entre diversos sistemas operacionais (Windows, Linux, AIX, etc.). Nele, estão presentes os processos de entrada, processamento e saída:

- Entrada: recebe as informações do cenário e coloca na memória;
- Processamento: executar a otimização do cenário;
- Saída: envia a melhor solução encontrada para o painel de planejamento.

Na execução do núcleo de otimização, que utiliza o AM para resolver o modelo proposto, é necessário obter um ótimo desempenho computacional. Para que isso seja possível, o núcleo de otimização dispõe de estruturas de dados e alocação de memória eficientes, bem como faz uso de processamento paralelo, onde o modelo de ilhas é utilizado.

No núcleo de otimização, as rotinas relativas ao controle evolutivo e a avaliação do indivíduo encontram-se em dois blocos distintos (Figura 10 e Figura 19), sendo que o bloco do controle evolutivo chama a rotina de avaliação para avaliar cada indivíduo da população. Além disso, podemos afirmar que o bloco do controle evolutivo é um AM sem customizações exclusivas para o FJSSP-LS, ou seja, a rotina poderia ser empregada em outro problema que não o de escalonamento de produção. Já a rotina de avaliação do indivíduo (que representa uma solução completa) foi desenvolvida para atender todos os requisitos do FJSSP-LS descritos anteriormente, considerando suas restrições e funcionalidades. A rotina de avaliação permite resolver também problemas menos complexos como o FJSSP (sem divisão de lotes) e o JSSP (*job shop* clássico), porém não poderia ser empregada em outro problema que não o de escalonamento.

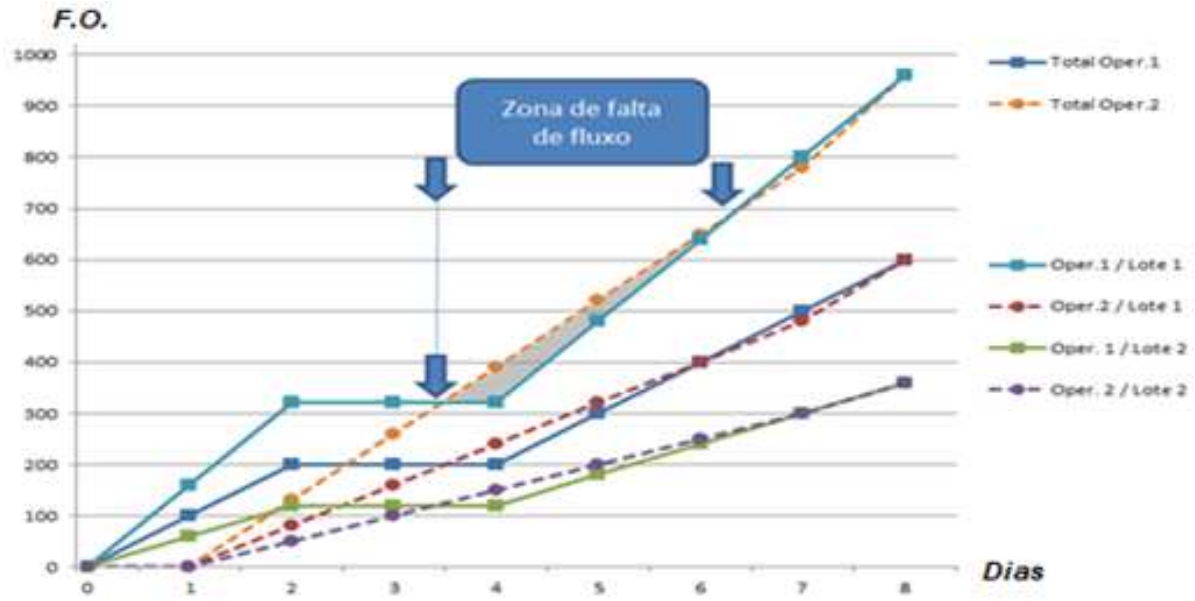
A implementação do núcleo de otimização é feita utilizando-se uma população de 50 indivíduos, onde a cada geração são selecionados aleatoriamente 14 indivíduos que, através de cruzamento e mutação, geram novos 700 indivíduos. Uma seleção é aplicada na população de 750 indivíduos, com a finalidade de manter novamente apenas 50 indivíduos que farão parte na geração seguinte. Estes valores seguem os mesmos valores propostos em (ZHAO et al., 2009).

4.1.1 Método de prevenção da falta de fluxo

A falta de fluxo ocorre quando um escalonamento é feito de forma que, durante a execução de uma operação, haja falta de peças devido à operação anterior ainda não ter finalizado uma quantidade suficiente de peças para abastecer a próxima operação. Neste caso, as máquinas da operação posterior ficam paradas por não terem condições de executar aquilo que havia sido programado.

A Figura 20 apresenta um exemplo de falta de fluxo, através de uma análise gráfica da das taxas de processamento de cada operação em cada máquina. Neste exemplo, a operação 1 é feita em duas máquinas, sendo que na primeira máquina, que processa o lote 1, a taxa de processamento é de 100 peças por dia e na segunda máquina, que processa o lote 2, é de 60 peças por dia. Além disso, as duas máquinas que executam a operação 1, não trabalham no intervalo dos dias 2 a 4 (representando, por exemplo, um fim-de-semana). A operação 2 também é feita em duas máquinas, sendo que na primeira máquina, que processa o lote 1, a taxa de processamento é de 80 peças por dia e na segunda máquina, que processa o lote 2, é de 50 peças por dia. Além disso, as duas máquinas que executam a operação 2 trabalham todos os dias. Todas as operações e lotes pertencem a uma mesma ordem de produção de 960 peças.

Figura 20: Exemplo de falta de fluxo

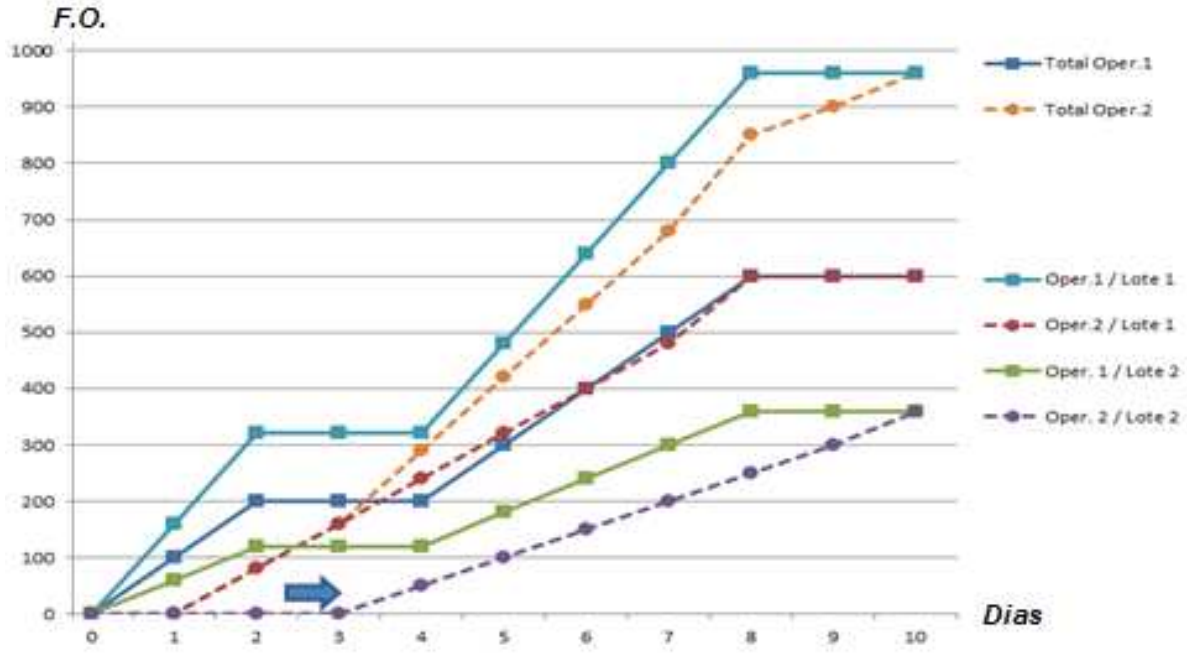


Fonte: Elaborado pelo autor.

No exemplo da Figura 20, cada uma das 4 linhas da parte inferior do gráfico representam os 2 lotes da operação 1 (linhas contínuas, azul e verde, com marcador quadrado) e os 2 lotes da operação 2 (linhas tracejadas, roxo e vermelho, com marcador redondo). Já as duas linhas da parte superior (linhas laranja e azul) representam a soma dos lotes da operação 1 e a soma dos lotes da operação 2. Dessa forma, a análise gráfica da falta de fluxo se torna bastante simples, pois temos apenas que verificar se a linha de total da operação 2 em algum momento cruza a linha de total da operação 1, indicando que haverá mais demanda do que oferta de peças. Esse intervalo, onde ocorre a falta de fluxo, está indicado no gráfico pelas setas que delimitam a “zona de falta de fluxo”, além de a região estar preenchida com uma cor mais escura.

A rotina que previne a falta de fluxo está implementada de maneira que, ao escalonar os lotes de operação 2, a taxa de processamento é verificada em cada intervalo (aqui no exemplo corresponde a 1 dia), verificando se ultrapassa o limite de peças já produzidas pelos lotes da operação 1. Caso ocorra essa falta de fluxo, o programa posterga o início da operação neste lote até uma posição onde não ocorra mais a falta de fluxo.

Figura 21: Exemplo de falta de fluxo solucionado



Fonte: Elaborado pelo autor.

Na Figura 21 é apresentado graficamente o resultado do ajuste necessário no escalonamento para que não ocorra a falta de fluxo. Pode-se verificar que a linha tracejada mais inferior (indicada com uma seta) tem seu início deslocado para o ponto 3, indicando que este lote deve ter uma postergação de 2 dias para que não haja falta de fluxo.

Em resumo, a rotina que previne a falta de fluxo implementa um procedimento análogo ao exemplificado graficamente acima, porém utilizando vetores em memória que representam os intervalos e quantidades das operações predecessora e sucessora dos lotes referentes a este par de operações. No caso de uma ordem ter O_i operações, serão verificados O_i-1 pares de operações predecessora e sucessora (1 e 2, 2 e 3, ..., O_{i-1} e O_i).

4.1.2 Estruturas e alocação de memória

Quando falamos em FJSSP-LS, deve-se ter em mente a diversidade de situações que podem compor um cenário de planejamento. Assim, podem existir cenários com muitas ordens de produção, muitas operações em cada ordem ou até mesmo várias máquinas alternativas para cada operação. Na modelagem do problema, verifica-se que são necessárias variáveis com múltiplas dimensões para possibilitar o mapeamento em memória das características do problema, ficando impossível de trabalhar com matrizes definidas estaticamente, pois seu tamanho excederá facilmente os limites de memória para esse fim. Dessa maneira, necessitar-se-á trabalhar com a alocação dinâmica da memória, o que não é nenhum problema em termos de programação, mas traz um *overhead* de processamento, pois nas variáveis estáticas os endereços de elemento de matriz são mais rapidamente acessados do que ponteiros de variáveis dinâmicas.

A criação de ponteiros individuais para cada elemento de dados acessado (como lotes, operações, ordens de produção, etc.), necessitam processamentos extras que deixam a execução do programa mais lento, além de que um número de ponteiros dinamicamente alocados aumenta ainda mais o consumo de memória. Para contornar esse problema, são definidas matrizes semelhantes ao modo estático, porém é criado um espaço de memória contínuo para todos os elementos da matriz, e não individualmente como normalmente é feito.

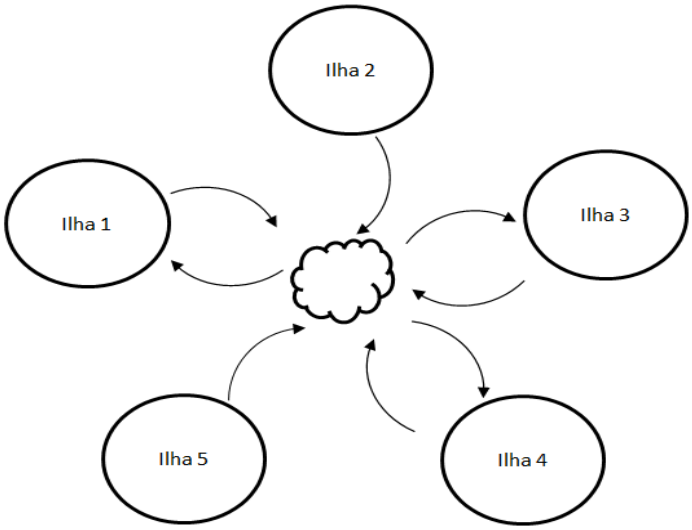
Com esses procedimentos adotados, as estruturas de dados em memória conseguem ser acessados sem comprometer o desempenho de processamento do algoritmo.

4.1.3 Paralelismo

Devido à complexidade computacional do FJSSP-LS, é necessário utilizar técnicas de implementação que permitam obter melhores soluções no menor tempo possível. Logo, o poder de processamento que estará disponível para o algoritmo será fator determinante em relação ao tempo de processamento necessário para obter boas soluções. Nesse contexto, a execução em paralelo tem participação fundamental no aumento do poder de processamento.

A ideia principal é ter várias instâncias do algoritmo sendo executadas em paralelo, de maneira independente e com suas próprias características. Para isso, será adotado na implementação o modelo de ilhas de processamento paralelo (CANTÚ-PAZ, 1998) com a topologia em estrela e sujeita ao efeito de correntes marítimas. A Figura 22 apresenta o modelo de ilhas proposto.

Figura 22: Modelo de ilhas



Fonte: Elaborado pelo autor.

Neste modelo, as ilhas representam execuções do algoritmo que evoluem de forma independente, cada uma com sua população. Inclui-se no modelo também o efeito migratório, onde indivíduos da população migram de uma ilha para outra através das correntes marinhas. Essas correntes têm duas características importantes: a primeira é a que determina se um indivíduo consegue sair da ilha e a segunda é a que determina se um indivíduo pode chegar na ilha. A motivação da implementação do efeito das correntes marinhas é possibilitar que as ilhas

tenham comportamentos diferentes, como manter-se mais isolada ou mais receptiva a imigrantes, além de permitir ou não que seus indivíduos influenciem em outras populações.

Cada ilha poderá ter uma configuração de parâmetros do algoritmo, tais como diferentes taxas de mutação e cruzamento, limite de gerações e reinicio do processo.

Para possibilitar esse modelo de processamento paralelo, são criadas *T+1* tarefas que são executadas em paralelo, sendo *T* tarefas destinadas as populações (ilhas) e uma destinada ao controle das correntes de migração e responsável pela verificação do critério de parada. A troca de informações entre as tarefas é feita uma vez por segundo, onde as *T* tarefas das ilhas enviam o melhor indivíduo para a tarefa principal e recebem de volta o melhor indivíduo global, exceto para 10% das ilhas que são configuradas para não receberem imigrantes.

4.1.4 Complexidade computacional

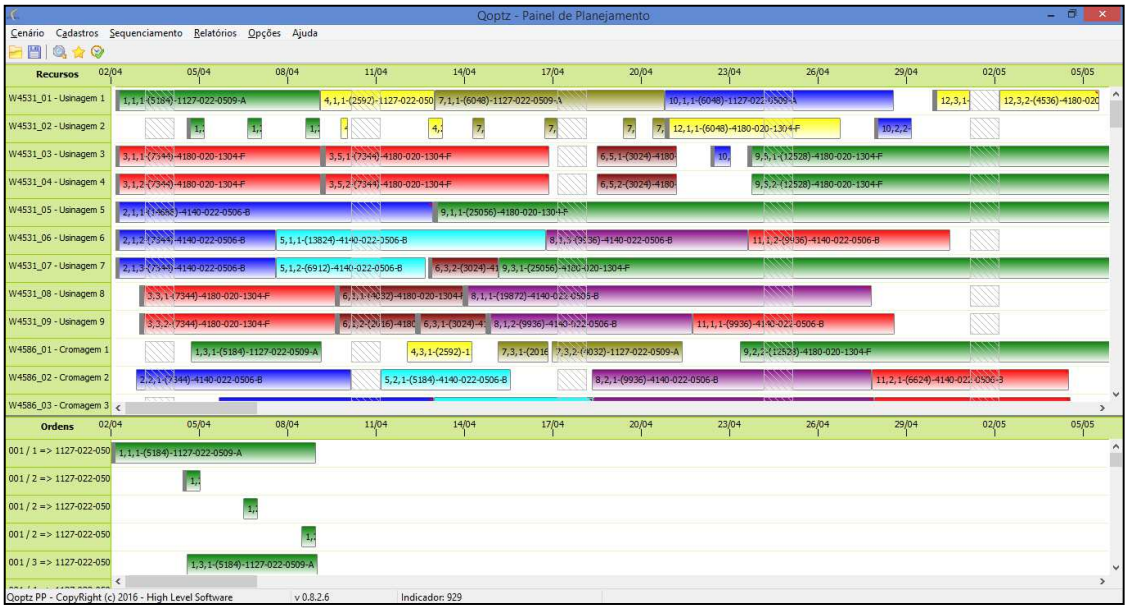
O AM implementado, que possibilita resolver o modelo matemático proposto, teve sua complexidade computacional calculada de maneira semelhante ao trabalho de (ZHAO, 2010), onde é apresentada a complexidade igual a $O(N_p * L_2^2)$, sendo N_p o tamanho da população utilizada e L_2 o tamanho do segundo cromossomo, onde está codificado a sequência em que as operações da ordens de produção devem ser escalonadas. No modelo proposto, L_2 terá seu tamanho determinado pelo número de ordens de produção, pelo número de operações de cada ordem e pelo número de lotes possíveis para cada operação. Já o número de lotes possíveis tem relação direta com a quantidade de máquinas alternativas em que cada operação poderá ser executada, de forma que quanto mais máquinas e ordens de produção tiver o cenário a ser otimizado, maior será a quantidade de instruções necessárias para resolvê-lo.

4.2 Painel de planejamento

A utilização de otimizadores de sequenciamento de produção requer ferramentas que permitam ao planejador de produção criar os cenários com os elementos do SMF, tais como máquinas, turnos, produtos e roteiros. Dessa forma, um cenário pode ser enviado ao núcleo de otimização de forma amigável, além de possibilitar a visualização gráfica, através de diagramas do tipo Gantt, dos resultados do escalonamento otimizado.

Para atender essas necessidades e possibilitar a validação das informações sequenciadas, foi criado um painel de planejamento, onde os cenários podem ser editados bem como os resultados analisados graficamente. Seu desenvolvimento foi feito na linguagem Pascal, utilizando a IDE do compilador Delphi 2007, que permite a criação de aplicações com interface gráfica de alta qualidade.

Figura 23: Painel de planejamento

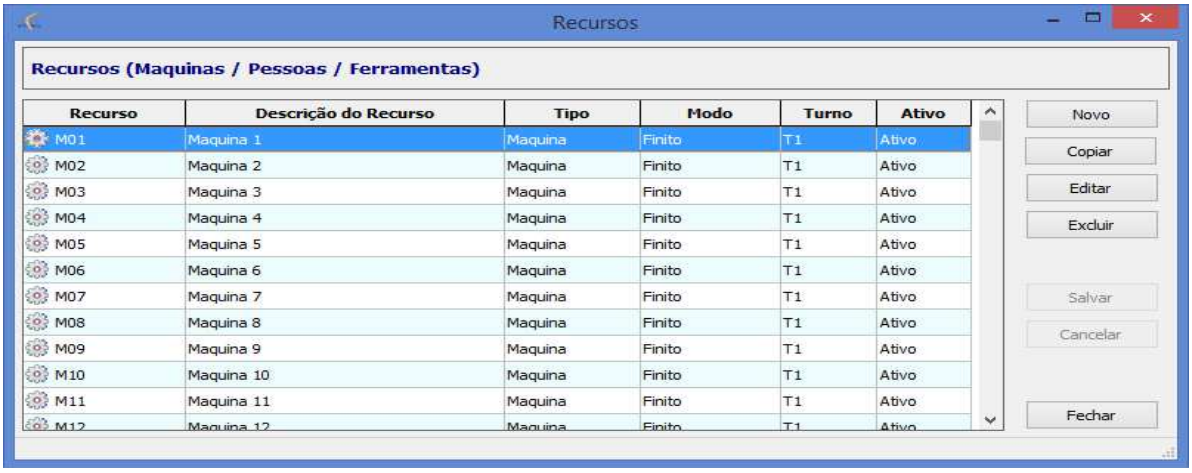


Fonte: Elaborado pelo autor.

A Figura 23 apresenta o painel de planejamento, onde a visualização gráfica dos lotes de produção é representada pelas barras horizontais. Na linha de cabeçalho é mostrada uma régua de tempo e na coluna à esquerda são mostradas as máquinas a que as linhas correspondem. Na parte inferior da tela, os lotes estão ordenados por ordem de produção e na primeira coluna é mostrada a ordem, operação e produto a que a linha corresponde. No topo da tela são mostrados os menus e os botões de funcionalidades, onde é possível criar, abrir e salvar um cenário, bem como executar a otimização do cenário, através da chamada do núcleo de otimização.

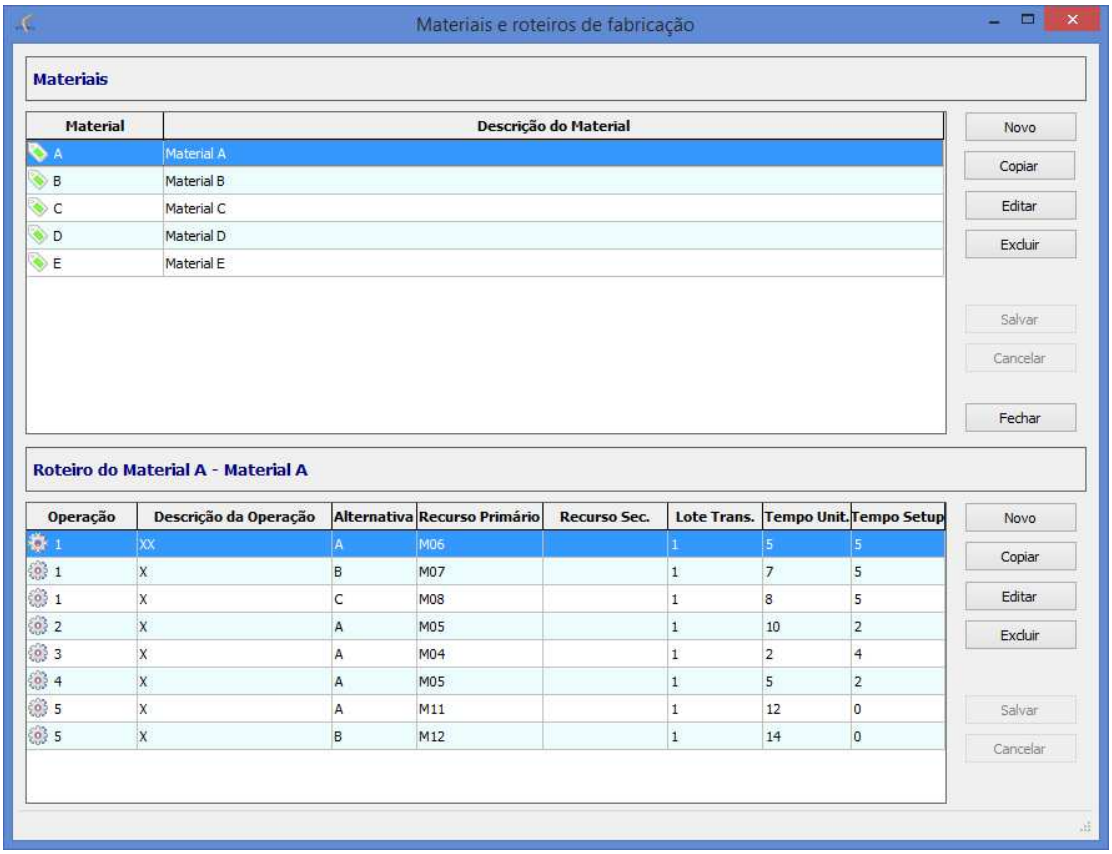
A manutenção das informações que compõem o cenário também pode ser acessada através dos menus do sistema, onde são apresentadas as informações dos recursos, produtos, roteiros de fabricação e ordens de produção, conforme pode ser visto na Figura 24, na Figura 25 e na Figura 26 respectivamente.

Figura 24: Cadastro de recursos do painel de planejamento



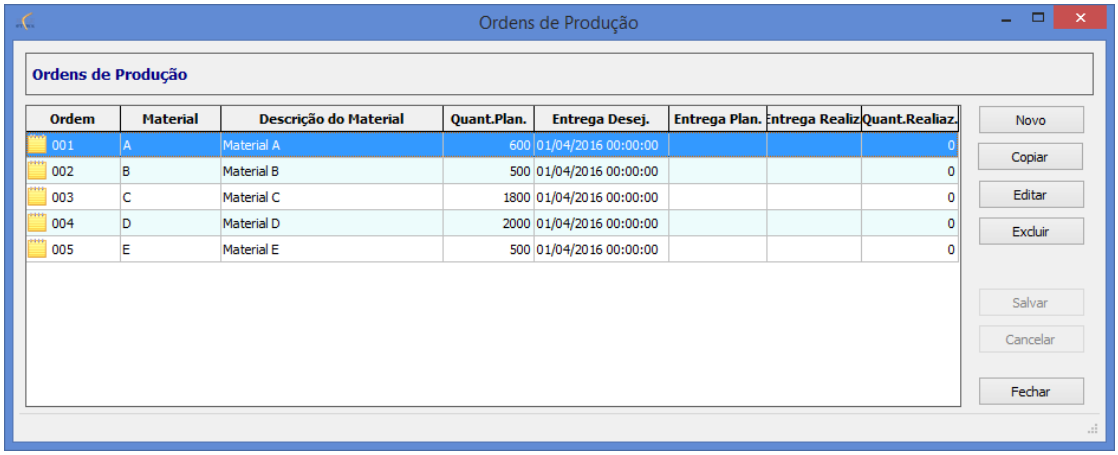
Fonte: Elaborado pelo autor.

Figura 25: Cadastro de produtos e roteiros de fabricação do painel de planejamento



Fonte: Elaborado pelo autor.

Figura 26: Cadastro de ordens de produção do painel de planejamento



Fonte: Elaborado pelo autor.

5 VALIDAÇÃO

Após a implementação do modelo, foi efetuada a validação da solução de otimização através da execução dos programas desenvolvidos, utilizando instâncias de testes que descrevem cenários de produção que devem ser otimizados.

A validação avaliou casos de testes em:

- Cenários unitários, onde é possível verificar o comportamento do otimizador em situações específicas, verificando detalhes do seu funcionamento;
- Cenários de JSSP clássicos, onde podemos efetuar a comparação com outros trabalhos;
- Cenários de FJSSP com divisão de lotes, onde podemos efetuar a comparação com outros trabalhos e também com a execução feita manualmente por planejadores de produção.

As figuras de escalonamento, apresentadas a seguir, são compostas por barras horizontais onde cada uma representa um lote escalonado, tendo em seu interior a indicação no formato: ordem, operação, lote - (quantidade) - produto (exemplo: “3,1,1-(4)-F”).

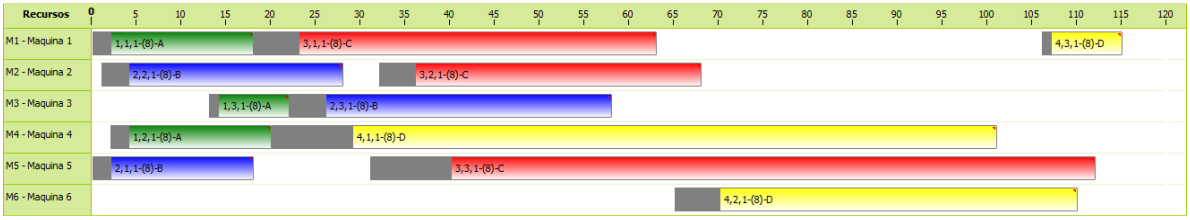
5.1 Avaliação de casos de teste unitários

Nesta etapa da avaliação foram verificadas situações específicas que possibilitam avaliar o comportamento do algoritmo implementado. Essas situações são representadas em um cenário pequeno (Tabela 5) com poucas máquinas e ordens, para que seja possível avaliar unitariamente a execução do programa. São testados os seguintes pontos:

- Verificação dos intervalos de indisponibilidade, onde atividades de setup e operações devem cruzar esses intervalos sem que contem no tempo da atividade;
- Verificação da alteração no escalonamento ao modificar os pesos que influenciam na função objetivo;
- Verificação do efeito do uso de lote de transferência;
- Verificação da consistência do fluxo de materiais, de modo que não haja falta de material entre operações.

A Figura 27 apresenta um escalonamento das ordens de produção onde foi otimizado apenas o tempo de *setup* ($P_2=1$, $P_1=0$, $P_3=0$). É possível verificar que a otimização funcionou corretamente, pois apesar da possibilidade da utilização de lotes o programa utilizou um sequenciamento das operações das ordens de produção que minimizou a necessidade de *setup*. Isso pode ser visto pela inexistência de lotes divididos, pois utilizar mais lotes acarretaria a necessidade de mais *setups*.

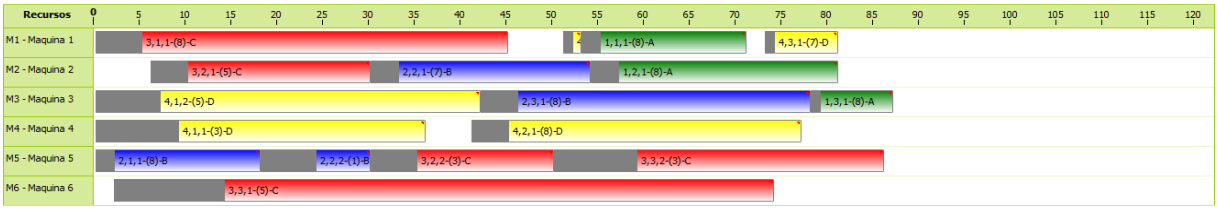
Figura 27: Escalonamento otimizado para setup



Fonte: Elaborado pelo autor.

Ainda na Figura 27, podemos verificar a correta utilização dos lotes de transferência, pois, por exemplo, o lote [1,2,1] inicia no tempo 4 e o lote [1,1,1] inicia no tempo 2. Essa diferença de tempo igual a 2 é o tempo necessário para que a primeira operação produza 1 peça e seja disponibilizada para a segunda operação.

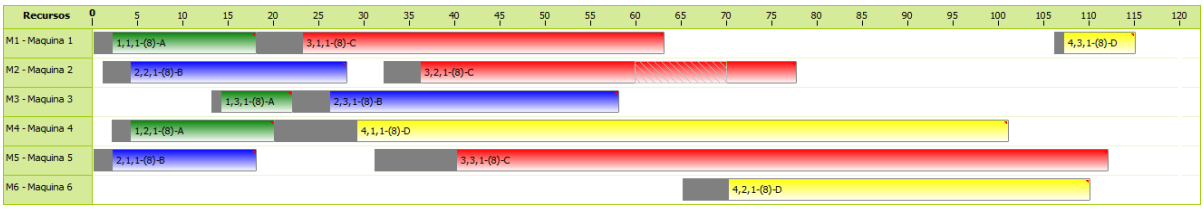
Figura 28: Escalonamento otimizado para makespan



Fonte: Elaborado pelo autor.

A Figura 28 apresenta um escalonamento das ordens de produção onde foi otimizado apenas o *makespan* ($P_3=1$, $P_1=0$, $P_2=0$). É possível verificar que a otimização funcionou corretamente, pois o programa utilizou um sequenciamento das operações das ordens de produção que procurou minimizar o tempo total de produção, utilizando para isso a divisão de lotes.

Figura 29: Escalonamento com período de indisponibilidade



Fonte: Elaborado pelo autor.

A Figura 29 apresenta um escalonamento das ordens de produção onde foi introduzido um período de indisponibilidade na máquina M2 no intervalo de tempo de 60 a 70. É possível verificar que o programa marcou o intervalo como indisponível e estendeu o lote [3,2,1] até o tempo 78 (na Figura 27 esse mesmo lote era finalizado no tempo 68). Também é possível observar que não há falta de fluxo de materiais para a operação posterior (lote [3,3,1]).

5.2 Comparação com problemas de escalonamento clássicos

Para a validação de problemas JSSP, utilizou-se duas instâncias de teste extensamente utilizadas na literatura: MT06 e MT10, originalmente propostos por (FISHER; THOMPSON,

1963) e posteriormente adaptado como FJSSP por (CHAMBERS; BARNES, 1996). Essas instâncias de testes são utilizadas em muitos trabalhos, contendo inclusive sua solução com valores ótimos.

Tabela 3: Instância de teste MT06

Job	**O_1**		**O_2**		**O_3**		**O_4**		**O_5**		**O_6**	
J_1	2	1	0	3	1	6	3	7	5	3	4	6
J_2	1	8	2	5	4	10	5	10	0	10	3	4
J_3	2	5	3	4	5	8	0	9	1	1	4	7
J_4	1	5	0	5	2	5	3	3	4	8	5	9
J_5	2	9	1	3	4	5	5	4	0	3	3	1
J_6	1	3	3	3	5	9	0	10	4	4	2	1

Fonte: FISHER; THOMPSON, 1963.

A Tabela 3 apresenta a máquina necessária e o tempo de processamento para cada operação O_j de cada ordem de produção J_i da instância MT06. Na tabela, cada linha representa uma ordem de produção, onde cada par de colunas indica o número da máquina necessária e o tempo de processamento unitário. Este cenário é composto por 6 ordens de produção, cada uma com 6 operações a serem alocadas em 6 máquinas diferentes.

Tabela 4: Instância de teste MT10

Job	**O_1**		**O_2**		**O_3**		**O_4**		**O_5**		**O_6**		**O_7**		**O_8**		**O_9**		**O_{10}**	
J_1	0	29	1	78	2	9	3	36	4	49	5	11	6	62	7	56	8	44	9	21
J_2	0	43	2	90	4	75	9	11	3	69	1	28	6	46	5	46	7	72	8	30
J_3	1	91	0	85	3	39	2	74	8	90	5	10	7	12	6	89	9	45	4	33
J_4	1	81	2	95	0	71	4	99	6	9	8	52	7	85	3	98	9	22	5	43
J_5	2	14	0	6	1	22	5	61	3	26	4	69	8	21	7	49	9	72	6	53
J_6	2	84	1	2	5	52	3	95	8	48	9	72	0	47	6	65	4	6	7	25
J_7	1	46	0	37	3	61	2	13	6	32	5	21	9	32	8	89	7	30	4	55
J_8	2	31	0	86	1	46	5	74	4	32	6	88	8	19	9	48	7	36	3	79
J_9	0	76	1	69	3	76	5	51	2	85	9	11	6	40	7	89	4	26	8	74
J_{10}	1	85	0	13	2	61	6	7	8	64	9	76	5	47	3	52	4	90	7	45

Fonte: FISHER; THOMPSON, 1963.

A Tabela 4 apresenta a máquina necessária e o tempo de processamento para cada operação O_j de cada ordem de produção J_i da instância MT10. Na tabela, cada linha representa uma ordem de produção, onde cada par de colunas indica o número da máquina necessária e o tempo de processamento unitário. Este cenário é composto por 10 ordens de produção, cada uma com 10 operações a serem alocadas em 10 máquinas diferentes.

As instâncias de testes utilizadas, MT06 e MT10, possuem um pequeno ajuste em relação aos problemas originais: a quantidade de cada ordem de produção é igual a 100 unidades cada, em vez de apenas 1 unidade como na versão original, seguindo o que foi proposto por (BUSCHER e SHEN, 2009), como o objetivo de possibilitar a divisão em lotes. Além disso,

nos resultados apresentados, as máquinas são numeradas partindo do índice 1 (M1) e não 0 como na tabela original.

5.3 Comparação com problemas de escalonamento com lotes

Para a validação de problemas JSSP-LS, são utilizadas as instâncias presentes em (ZHAO et al., 2010). São três problemas que utilizam a divisão de lotes e apresentam os resultados, sendo possível a comparação entre os trabalhos. Nestes cenários, podemos observar que cada operação pode ser executada em mais de uma máquina, ou seja, existe um conjunto de máquinas alternativas para a operação.

A primeira instância utilizada é um cenário com 4 ordens de produção, cada uma de um produto diferente e que possui seu próprio roteiro de fabricação, onde existem 6 máquinas disponíveis para executar as operações.

Tabela 5: Instância de teste 4x6

Job	Operation	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	Machine 6
	1	2	3	4	–	–	–
J_1	2	–	3	–	2	4	–
	3	1	4	5	–	–	–
	1	3	–	5	–	2	–
J_2	2	4	3	–	–	6	–
	3	–	–	4	–	7	11
	1	5	6	–	–	–	–
J_3	2	–	4	–	3	5	–
	3	–	–	13	–	9	12
	1	9	–	7	9	–	–
J_4	2	–	6	–	4	–	5
	3	1	–	3	–	–	3

Fonte: ZHAO et al., 2010.

A Tabela 5 apresenta o roteiro de fabricação de cada produto do primeiro problema, indicando as operações de cada produto, bem como as máquinas alternativas e seus respectivos tempos. O tempo de *setup* equivale ao tempo de uma unidade e o lote de transferência é igual a 1. Neste cenário, são escalonadas 4 ordens de produção, sendo uma para cada produto e todas com quantidade igual a 8 peças.

A segunda instância utilizada é um cenário com 6 ordens de produção, cada uma de um produto diferente e que possui seu próprio roteiro de fabricação, onde existem 6 máquinas disponíveis para executar as operações.

Tabela 6: Instância de teste 6x6

Job	Operation	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	Machine 6
J_1	1	2/1	–	–	–	–	–
	2	–	–	3/2	2/1	–	–
	3	–	2/1	–	2/2	3/2	2/1
	4	–	5/3	–	6/2	–	–
	5	–	–	2/1	–	–	2/1
	6	–	1/1	–	–	1/1	–
J_2	1	–	2/1	–	1/1	–	–
	2	–	–	4/2	–	–	–
	3	8/2	–	–	–	7/2	7/3
	4	–	4/2	5/1	5/2	–	–
	5	–	–	1/1	–	–	1/1
	6	–	4/2	–	–	5/1	–
J_3	1	4/2	–	5/2	–	–	–
	2	–	5/3	–	5/2	–	–
	3	–	–	1/1	–	1/1	–
	4	–	6/3	–	–	7/2	–
	5	–	2/2	2/1	–	–	3/1
J_4	1	4/2	–	–	4/1	–	–
	2	–	–	2/1	–	–	–
	3	–	4/1	–	3/1	–	3/1
	4	–	–	6/2	–	5/2	–
	5	6/1	–	–	–	–	–
J_5	1	2/1	–	–	3/1	–	–
	2	–	5/1	–	–	4/1	–
	3	–	–	1/1	1/1	–	–
	4	–	–	3/1	–	–	2/1
	5	–	3/1	2/2	–	–	–
	6	–	–	–	–	2/2	–
J_6	1	2/1	–	3/1	–	2/2	–
	2	–	4/1	–	–	3/2	–
	3	–	–	–	6/2	–	6/1
	4	–	2/1	–	2/1	–	–
	5	–	–	1/2	–	–	–
	6	2/2	–	–	3/1	2/2	–

Fonte: ZHAO et al., 2010.

A Tabela 6 apresenta o roteiro de fabricação de cada produto do segundo problema, indicando as operações de cada produto, bem como as máquinas alternativas e seus respectivos tempos de produção, seguido do tempo de *setup*. O lote de transferência é igual a 1. Neste cenário, são escalonadas 6 ordens de produção, sendo uma para cada produto e todas com quantidade igual a 10 peças.

A terceira instância utilizada é um cenário com 6 ordens de produção, cada uma de um produto diferente que possui seu próprio roteiro de fabricação, onde existem 12 máquinas disponíveis para executar as operações. Segundo ZHAO et al. (2010), este cenário é oriundo de uma situação real proposta em um *workshop* em que os autores participaram.

Tabela 7: Instância de teste 6x12

Job	Original batch size		Operation	Unit processing time / set-up time for operations on alternative machines											s
				E1	*E2*	*E3*	*E4*	*E5*	*E6*	*E7*	*E8*	*E9*	*E10*	*E11*	*E12*
Cloth-edged paper cone (J_1)	600	1	Coat	–	–	–	–	–	5/5	7/5	8/5	–	–	–	–
		2	Stuff	–	–	–	–	10/2	–	–	–	–	–	–	–
		3	Heat-compress	–	–	–	2/4	–	–	–	–	–	–	–	–
		4	Die cut	–	–	–	–	5/2	–	–	–	–	–	–	–
		5	Final check	–	–	–	–	–	–	–	–	–	–	12/0	14/0
Foam-edged paper cone (J_2)	500	1	Paint internal circle	–	–	–	–	–	5/5	9/4	6/7	–	–	–	–
		2	Interim check	–	–	–	–	–	–	–	–	3/0	4/0	–	–
		3	Built paper cone	–	4/8	4/3	–	–	–	–	–	–	–	–	–
		4	Seal obversely	–	15/4	7/7	–	–	–	–	–	–	–	–	–
		5	Seal reversely	–	5/3	5/3	–	–	–	–	–	–	–	–	–
		6	Final check	–	–	–	–	–	–	–	–	–	–	10/0	8/0
Rubber-edged paper cone (J_3)	1 800	1	Paint internal circle	–	–	–	–	–	5/4	7/6	8/6	–	–	–	–
		2	Coat	–	–	–	–	–	6/6	6/8	10/6	–	–	–	–
		3	Interim check	–	–	–	–	–	–	–	–	4/0	5/0	–	–
		4	Built paper cone	–	15/5	14/3	–	–	–	–	–	–	–	–	–
		5	Seal reversely	–	5/3	3/1	–	–	–	–	–	–	–	–	–
		6	Final check	–	–	–	–	–	–	–	–	–	–	10/0	11/0
Paper sub-cone (J_4)	2 000	1	Make brass net	–	–	–	–	6/2	–	–	–	–	–	–	–
		2	Shape up with pulp	4/2	–	–	–	–	–	–	–	–	–	–	–
		3	Interim check	–	–	–	–	–	–	–	–	3/0	5/0	–	–
		4	Die cut	–	–	–	–	5/3	–	–	–	–	–	–	–
		5	Final check	–	–	–	–	–	–	–	–	–	–	6/0	9/0
Paper cap (J_5)	500	1	Paint internal circle	–	–	–	–	–	5/4	6/4	6/5	–	–	–	–
		2	Coat	–	–	–	–	–	5/6	5/6	6/6	–	–	–	–
		3	Interim check	–	–	–	–	–	–	–	–	5/0	5/0	–	–
		4	Heat-compress	–	–	–	6/4	–	–	–	–	–	–	–	–
		5	Seal obversely	–	9/2	10/4	–	–	–	–	–	–	–	–	–
		6	Seal reversely	–	5/2	4/2	–	–	–	–	–	–	–	–	–
		7	Final check	–	–	–	–	–	–	–	–	–	–	9/0	10/0

Fonte: ZHAO et al., 2010.

A Tabela 7 apresenta o roteiro de fabricação de cada produto do terceiro cenário, indicando as operações de cada produto, bem como as máquinas alternativas e seus respectivos tempos de produção e *setup*. O lote de transferência é igual a 1. Neste cenário, são escalonadas 5 ordens de produção, sendo uma para cada produto e todas com as quantidades indicadas na tabela.

Para que a comparação dos resultados destes três cenários de escalonamento com lotes com resultados de outros trabalhos seja possível, foi utilizado novamente no cálculo da função objetivo somente o *makespan*, ou seja, utilizando os parâmetros *P1* e *P2* iguais a zero e *P3* igual a 1.

Além desses três cenários, foi utilizada mais uma instância, que é um cenário obtido de uma situação real de uma empresa que trabalha com produção de peças em um SMF. Este cenário é composto de 12 ordens de produção que utilizam 3 roteiros de fabricação distintos (3 produtos diferentes), 19 máquinas e 22 ferramentas, além de conter intervalos de indisponibilidade originários dos turnos de trabalho.

Tabela 8: Roteiros de fabricação da instância de teste 12x41

Produto	**Roteiro de Fabricação**					
	Operação	***Lote Transf.***	***Máquina***	***Ferramenta***	***Tempo (s)***	***Setup (s)***
A	1	24	M01..M09	F20	97	10800
A	2	24	M01..M09	F21	25	10800
A	3	48	M10..M16	F22	74	0
A	4	864	M17..M19	F23	36	3600

Produto	Roteiro de Fabricação					
	Operação	*Lote Transf.*	*Máquina*	*Ferramenta*	*Tempo (s)*	*Setup (s)*
B	1	24	M01..M09	F24..F26	51	10800
B	2	48	M10..M16	F27..F30	74	0
B	3	864	M17..M19	F31..F33	33	3600
C	1	24	M01..M09	F34..F35	69	10800
C	2	48	M10..M16	F36..F37	74	0
C	3	24	M01..M09	F38..F39	65	10800
C	4	24	M17..M19	F31..F33	32	3600
C	5	864	M01..M09	F40..F41	78	10800

Fonte: Elaborado pelo autor.

A Tabela 8 apresenta os tempos de produção e de *setup* para cada operação de cada roteiro de fabricação dos produtos. Na tabela, são informados também em qual grupo de máquinas e qual grupo de ferramentas é possível executar a operação, onde, por exemplo, M01..M09 indica que a operação pode ser executada em qualquer uma das nove máquinas de número 01 a 09. Além disso, o lote de transferência de cada operação é informado. Neste cenário temos a aplicação de máquinas alternativas e de recursos secundários, onde além de uma máquina para executar a operação, é necessário também uma ferramenta.

Tabela 9: Ordens de produção da instância de teste 12x41

Ordem	*Produto*	*Quantidade*	*Prazo de Entrega (h)*
1	A	5.184	216
2	B	29.376	216
3	B	14.688	216
4	A	2.592	384
5	B	20.736	384
6	C	6.048	384
7	A	6.048	552
8	B	39.744	552
9	C	25.056	552
10	A	6.048	720
11	B	19.872	720
12	C	6.048	720

Fonte: Elaborado pelo autor.

A Tabela 9 apresenta as 12 ordens de produção que devem ser consideradas no escalonamento da instância 12x41. As colunas representam o número da ordem de produção, o produto, a quantidade a ser produzida e o prazo de entrega desejado. Assim, cruzando as informações da Tabela 8 com os dados da Tabela 9, montamos o cenário completo 12x41.

Na otimização deste cenário, foi utilizado no cálculo da função objetivo somente o tempo de atraso, ou seja, utilizou-se os parâmetros P_2 e P_3 iguais a zero e P_1 igual a 1. Assim, sempre que uma ordem de produção for escalonada de modo que sua última operação termine após o prazo de entrega, estará configurado um atraso, calculado como a diferença entre o tempo final da ordem e o tempo desejado de entrega (prazo de entrega).

5.4 Análise dos parâmetros de exploração e intensificação

Os parâmetros que orientam o caráter exploratório do AM devem ser analisados para que possam levar a uma busca ampla de lugares ainda não explorados do espaço de busca. No modelo desenvolvido, o principal parâmetro que controla o caráter exploratório do AM é a taxa de mutação, para a qual foram feitas várias simulações que não indicaram um valor ideal para essa taxa. Desse modo, optou-se por uma taxa de cruzamento variável ente 0 e 0,1 e que é definida aleatoriamente a cada 50 gerações. Outro importante parâmetro do AM é a taxa de cruzamento, onde optou-se também por uma taxa variável entre 0 e 0,8 e que é definida aleatoriamente a cada 50 gerações

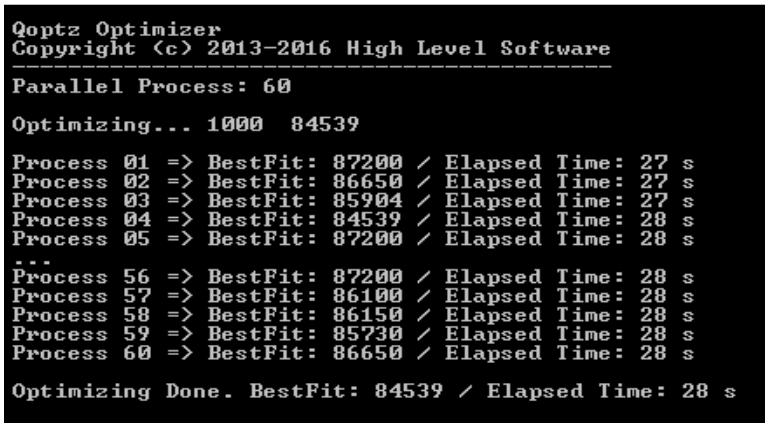
A busca local proporciona uma melhora nos indivíduos da população através do auto aprendizado desses indivíduos. Dessa forma, intensificamos a busca em determinada região do espaço de busca, onde o melhor indivíduo da população e mais dois outros indivíduos escolhidos aleatoriamente são submetidos ao processo de busca local em que 10% de seus genes são sorteados aleatoriamente para a execução do método de inserção. O número de indivíduos selecionados para a busca local e a quantidade de genes processados leva em consideração que cada ciclo do método de inserção requer a execução da rotina de avaliação para o cálculo da função objetivo, o que aumenta significativamente o tempo de processamento. Logo, é inviável aplicar a busca local em todos os indivíduos e para todos seus genes.

6 RESULTADOS OBTIDOS

A solução implementada possibilita a resolução de cenários de escalonamento de produção, buscando otimizá-los através da minimização da função objetivo, composta por suas variáveis de decisão e pesos associados.

O AM capaz de trabalhar com o modelo proposto está codificado em linguagem C e cada instância foi executada 100 vezes, com um critério de parada de 1000 gerações. Para as execuções cujos resultados foram utilizados neste capítulo, foi utilizado um computador com sistema operacional Windows 8.1 e processador Intel Core i7-4500U 2.4GHz e 8GB de memória, onde somente um núcleo físico de processamento foi utilizado exclusivamente para a execução (sem processamento paralelo). Para a validação do paralelismo utilizado no programa desenvolvido, foi utilizada uma plataforma de hardware baseada no coprocessador Intel Xeon Phi 5110P com 60 núcleos de 1.05GHz e 32GB de memória (com processamento paralelo), cujo resultado comprovou o correto funcionamento do modelo de ilhas. A Figura 30 apresenta o resultado da execução do AM utilizando processamento paralelo.

Figura 30: Execução do AM utilizando processamento paralelo



Fonte: Elaborado pelo autor.

Na apresentação dos resultados dos cenários otimizados, são mostrados a seguir a representação gráfica do escalonamento otimizado através de um gráfico de Gantt, onde as partições escuras indicam tempos de *setup* utilizados e as partições coloridas o processamento das operações. Em cada barra é apresentado uma indicação no formato: ordem de produção, operação, lote - (quantidade) - produto (exemplo: “3,1,1-(4)-F”).

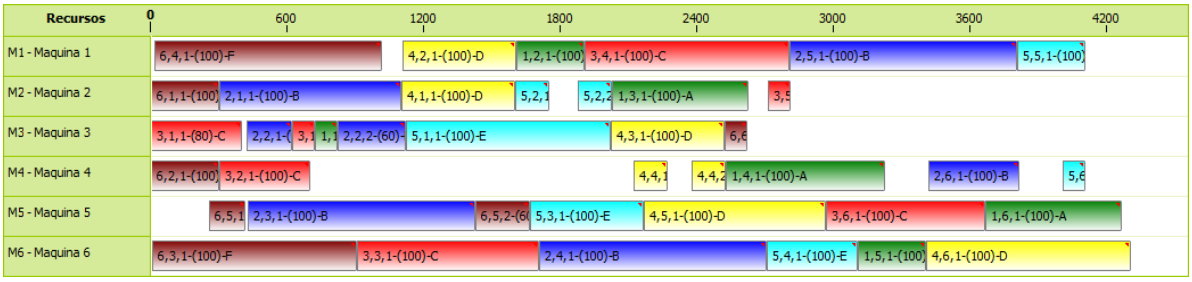
6.1 Resultados da instância MT06

A instância MT06, composta por 6 ordens de produção e 6 máquinas, foi processada pelo programa desenvolvido e a otimização obtida do *makespan* é descrita abaixo:

- Melhor resultado: 4306
- Pior resultado: 4363
- Média dos resultados: 4323,4
- Desvio padrão: 21,65
- Media de gerações p/melhor resultado: 542
- Tempo médio de execução: 32 seg.

O melhor resultado encontrado é apresentado na Figura 31, onde é possível verificar o excelente aproveitamento da máquina M6, que é a mais sobrecarregada.

Figura 31: Gráfico do melhor resultado da instância MT06



Fonte: Elaborado pelo autor.

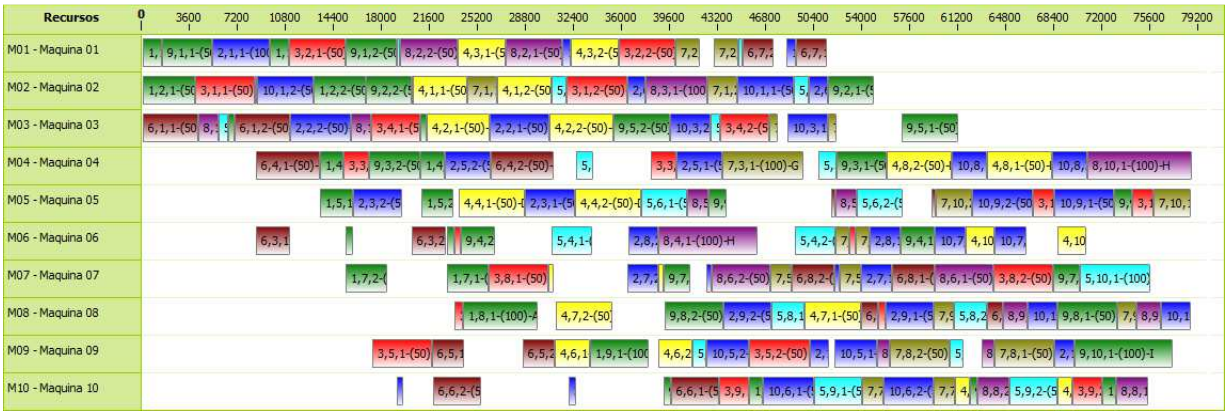
6.2 Resultados da instância MT10

A instância MT10, composta por 10 ordens de produção e 10 máquinas, foi processada pelo programa desenvolvido e a otimização obtida do *makespan* é descrita abaixo:

- Melhor resultado: 78712
- Pior resultado: 80548
- Média dos resultados: 79776
- Desvio padrão: 760,81
- Media de gerações p/melhor resultado: 632
- Tempo médio de execução: 82 seg.

O melhor resultado encontrado é apresentado na Figura 32, onde é possível verificar o bom agrupamento dos blocos nas máquinas M01 a M04 e nas máquinas M07 a M10, mas alguns desencaixes nas máquinas M05 e M06.

Figura 32: Gráfico do melhor resultado da instância MT10



Fonte: Elaborado pelo autor.

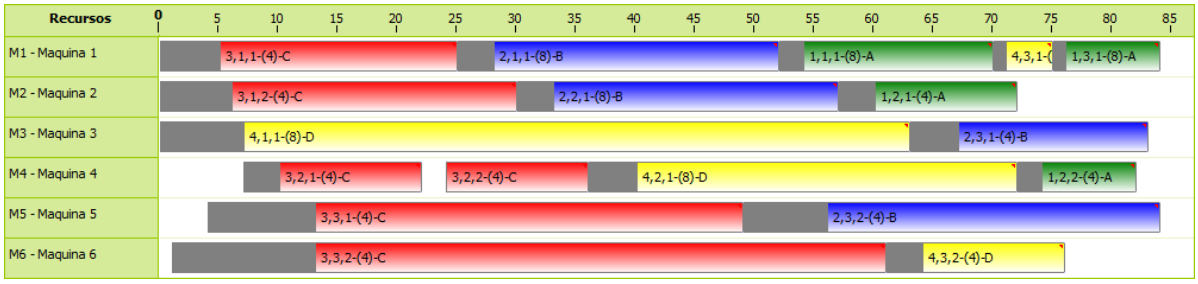
6.3 Resultados da instância 4x6

A instância 4x6, composta por 4 ordens de produção e 6 máquinas, foi processada pelo programa desenvolvido e a otimização obtida do *makespan* é descrita abaixo:

- Melhor resultado: 84
- Pior resultado: 87
- Média dos resultados: 85,3
- Desvio padrão: 1,16
- Media de gerações p/melhor resultado: 425
- Tempo médio de execução: 16 seg.

O melhor resultado encontrado é apresentado na Figura 33, onde é possível verificar o excelente aproveitamento da divisão em lotes da operação 2 da ordem 3.

Figura 33: Gráfico do melhor resultado da instância 4x6



Fonte: Elaborado pelo autor.

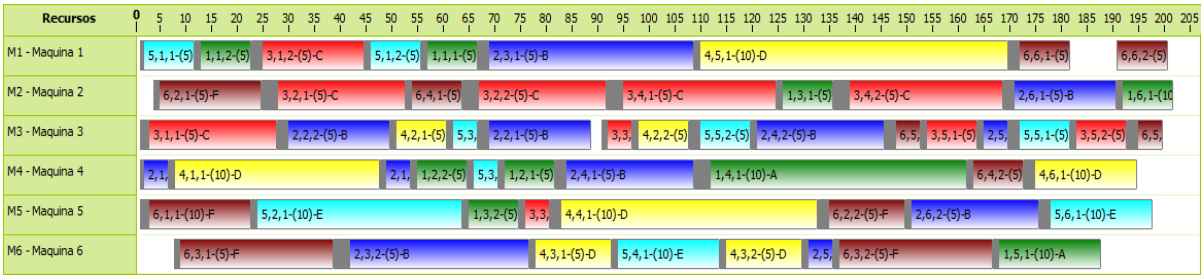
6.4 Resultados da instância 6x6

A instância 6x6, composta por 6 ordens de produção e 6 máquinas, foi processada pelo programa desenvolvido e a otimização obtida do *makespan* é descrita abaixo:

- Melhor resultado: 201
- Pior resultado: 210
- Média dos resultados: 205,1
- Desvio padrão: 3,60
- Media de gerações p/melhor resultado: 309
- Tempo médio de execução: 36 seg.

O melhor resultado encontrado é apresentado na Figura 34, onde é possível verificar o excelente aproveitamento da divisão em lotes através de sua distribuição uniforme em todas as máquinas.

Figura 34: Gráfico do melhor resultado da instância 6x6



Fonte: Elaborado pelo autor.

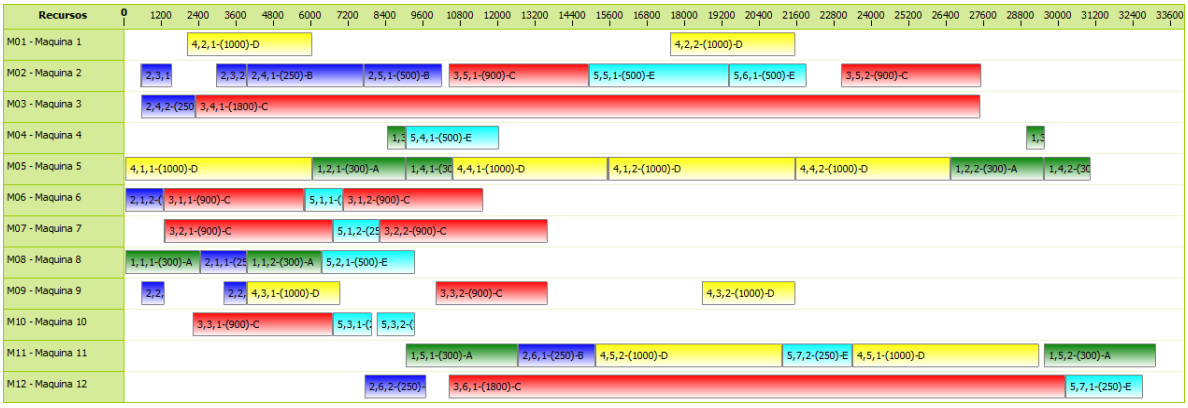
6.5 Resultados da instância 5x12

A instância 5x12, composta por 5 ordens de produção e 12 máquinas, foi processada pelo programa desenvolvido e a otimização obtida do *makespan* é descrita abaixo:

- Melhor resultado: 33118
- Pior resultado: 33439
- Média dos resultados: 33281
- Desvio padrão: 131,98
- Media de gerações p/melhor resultado: 372
- Tempo médio de execução: 33 seg.

O melhor resultado encontrado é apresentado na Figura 35, onde é possível verificar o excelente aproveitamento da máquina 5, que é a mais sobrecarregada.

Figura 35: Gráfico do melhor resultado da instância 5x12



Fonte: Elaborado pelo autor.

6.6 Resultados da instância 12x41

A instância 12x41, composta por 12 ordens de produção de 41 máquinas, foi processada pelo programa desenvolvido e a otimização obtida do atraso é descrita abaixo:

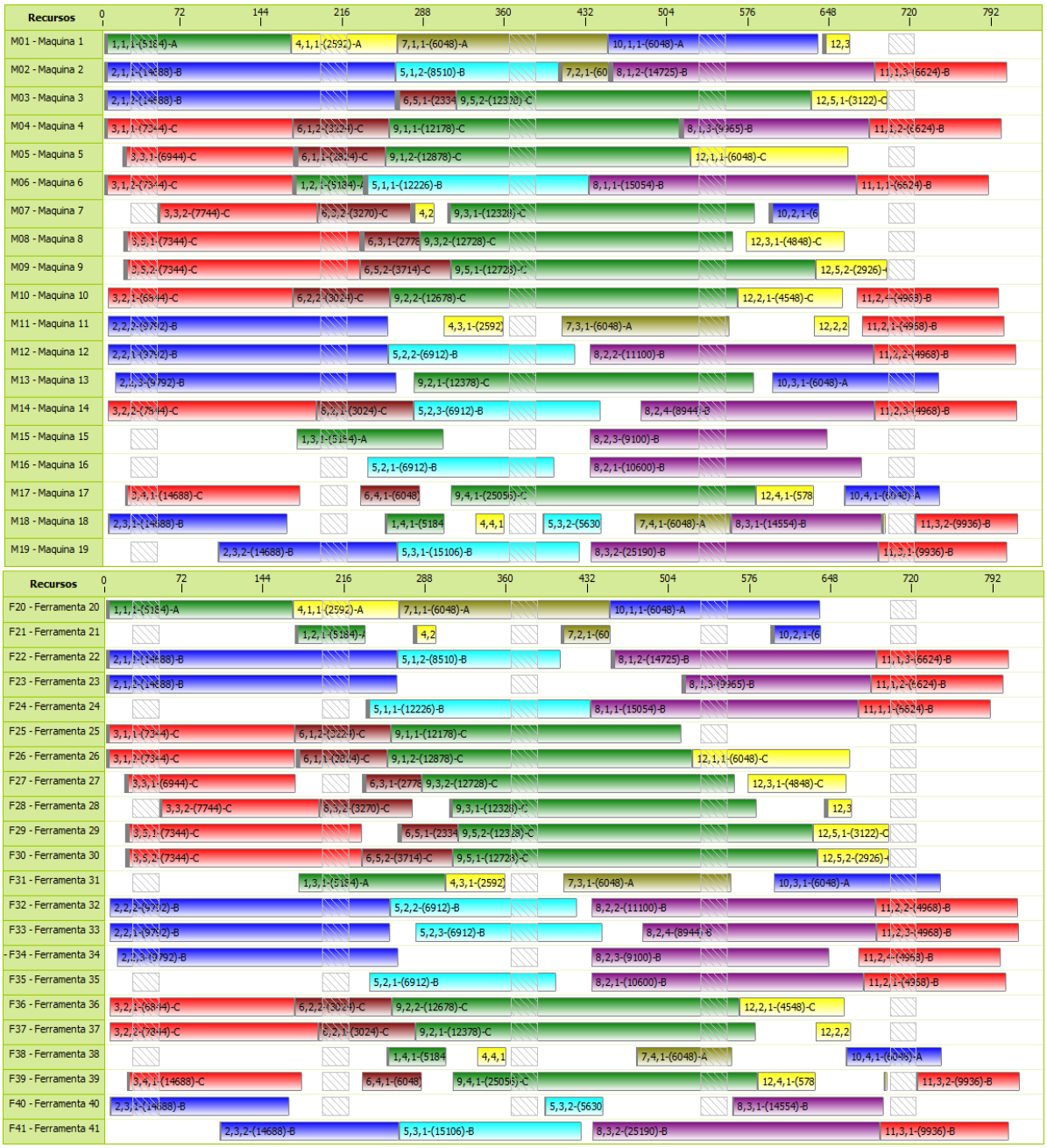
- Melhor resultado: 528
- Pior resultado: 630
- Média dos resultados: 561,6
- Desvio padrão: 37,08
- Media de gerações p/melhor resultado: 247
- Tempo médio de execução: 58 seg.

O melhor resultado encontrado é apresentado na Figura 36, onde as partições em cinza indicam tempos de *setup* utilizados.

As áreas com linhas diagonais representam os intervalos de indisponibilidade e na figura pode ser verificada uma divisão onde a parte superior são as máquinas (recursos principais) e a parte inferior são as ferramentas (recursos secundários).

A instância 12x41 foi apresentada a um grupo de seis especialistas em planejamento de produção de uma empresa da indústria metalmecânica e a eles foi solicitado que executassem o melhor sequenciamento possível. Após utilizarem 8 horas de trabalho no problema proposto, o melhor resultado obtido manualmente foi 653.

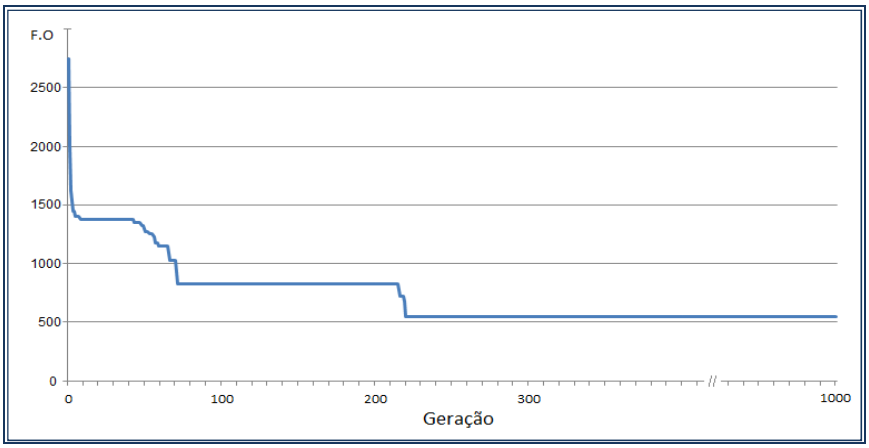
Figura 36: Gráfico do melhor resultado da instância 12x41



Fonte: Elaborado pelo autor.

A Figura 37 apresenta a evolução da função objetivo à medida que as gerações são processadas na instância 12x41. Pode-se notar que continuamente a solução é melhorada até um determinado limiar onde se estabiliza até o final das 1000 gerações. O ponto de estabilidade está em concordância com a média de gerações em que o programa localiza o melhor resultado que foi de 247 gerações.

Figura 37: Evolução da solução ao longo das gerações



Fonte: Elaborado pelo autor.

6.7 Comparativo dos resultados com outros trabalhos

Vários cenários do JSSP-LS foram otimizados com o AM implementado e seus resultados apresentados em detalhes nas seções anteriores. A Tabela 10 apresenta um resumo dos resultados obtidos, comparando-os com outros trabalhos pesquisados na literatura existente.

Tabela 10: Comparativo de resultados

Instância	Resultado Obtido	Resultado Comparado	% Melhora	Trabalho Comparado
MT06	4306	4330	5,5 %	BUSCHER e SHEN, 2009
MT10	78712	63930	-23,1 %	BUSCHER e SHEN, 2009
4x6	84	85	1,2 %	ZHAO et al., 2010
6x6	202	213	5,2 %	ZHAO et al., 2010
5x12	33118	43256	23,4 %	ZHAO et al., 2010
12x41	528	653	19,1 %	Escalonamento manual

Fonte: Elaborado pelo autor.

Em 5 dos 6 cenários otimizados foi obtido uma melhora significativa em relação aos trabalhos utilizados como comparação. Mesmo na instância 4x6 onde a melhora foi “apenas” 1 unidade de tempo, note-se que em dois trabalhos (ZHAO et al., 2009) e (ZHAO et al., 2010), os autores não foram além do resultado de 85, demonstrando a real dificuldade em se obter resultados abaixo dessa marca. Semelhante resultado foi obtido na instância 6x6, com uma melhora de 5,2%.

Em (BUSCHER e SHEN, 2009), podemos observar que o LB (*Lower Bound*) da instância MT06 é 4300. Ou seja, nenhum resultado pode ser conseguido abaixo deste valor. Em verdade, o valor de 4306 é um resultado ótimo para a instância MT06 com divisão de lotes, pois adiciona apenas 6 unidades de tempo ao LB, referentes ao mínimo possível para executar operações predecessoras fora da máquina que define o LB.

Nas instâncias 5x12 e 12x41, cuja origem é uma situação real de produção, os resultados foram amplamente satisfatórios, com melhora de 23,4% e 19,1% respectivamente. Esses resultados são especialmente importantes, pois comprovam a eficiência do AM implementado para cenários reais.

Apenas em um dos cenários não foi obtido um resultado melhor do que o trabalho comparado. Apesar da otimização feita pelo AM ter sido executada de maneira correta e eficiente, o resultado atingido por BUSCHER e SHEN (2009) na instância MT10 é significativamente melhor do que o resultado do AM.

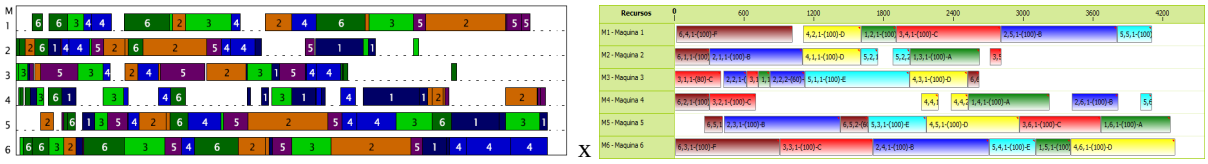
7 CONCLUSÃO

Este trabalho propôs e implementou um modelo de otimização de escalonamento de ordens de produção utilizando lotes de tamanho variável, que considera a utilização de máquinas alternativas, bem como a utilização de recursos secundários, a observação de intervalos de indisponibilidade e de lotes de transferência. É utilizado um AM, onde um cromossomo representa o tamanho dos lotes e o outro a ordem de sequenciamento. Instâncias do JSSP e do FJSSP foram utilizadas nos testes e os resultados obtidos comprovam que o algoritmo proposto consegue otimizar o escalonamento das ordens de produção de cada instância de maneira eficiente.

A consideração do tamanho de lote variável é um ponto fundamental para muitas situações num SMF e, nesses casos, não considerá-lo no modelo a ser utilizado pode gerar resultados pouco aderentes no momento da execução do sequenciamento no chão de fábrica. O modelo proposto apresenta a flexibilidade necessária para mapear, tanto problemas de testes descritos na literatura, quanto cenários reais de produção, permitindo refletir a realidade fabril no escalonamento otimizado.

Uma importante contribuição deste trabalho foi a utilização de lotes de transferência no modelo, o que proporcionou um resultado otimizado com um número menor de lotes subdivididos, conforme pode ser visto na Figura 38, onde em o resultado de BUSCHER e SHEN (2009) utilizou um total de 135 lotes divididos, enquanto com a utilização de lotes de transferência foram necessários apenas 41 lotes. A importância de um modelo apresentar resultados com um menor número de divisões está no fato de que, em SMFs reais, existe um custo de *setup* relacionado à troca de produto a ser processado na máquina. Assim, um menor número de lotes divididos faz com que haja uma necessidade menor de troca de tarefas executadas na máquina, diminuindo o tempo e o custo de *setup*.

Figura 38: Utilização de lotes de transferência



Fonte: BUSCHER e SCHEN, 2009 x Elaborado pelo autor.

O resultado da otimização da instância MT06 superou os resultados apresentados em (BUSCHER e SHEN, 2009) obtendo um valor ótimo, o que já não ocorreu com a instância MT10, onde o resultado foi inferior ao referido trabalho.

O resultado da otimização da instância 4x6 superou os resultados apresentados em trabalhos anteriores: (ZHAO et al., 2009) e (ZHAO et al., 2010). A dimensão reduzida da instância 4x6 permite uma análise lote a lote do resultado obtido, facilitando seu entendimento. Na instância 6x6 também foi superado o resultado dos trabalhos de ZHAO, citados anteriormente, bem como na instância 5x12, onde a melhora foi muito significativa.

O resultado da otimização para a instância 12x41 proposta mostrou-se significativamente melhor do que o sequenciamento manual feito por especialistas em planejamento de produção. Além disso, os especialistas necessitaram 8 horas para chegar em seu melhor resultado, um tempo muito maior do que o programa que obteve suas soluções em menos de 1 minuto.

Os resultados dos cenários otimizados confirmam a validade do modelo proposto para a otimização do escalonamento de ordens de produção com divisão em lotes, bem como atingiram uma excelente performance em todas as instâncias testadas.

7.1 Trabalhos futuros

A proposta do presente trabalho foi desenvolver uma solução que fosse aderente à realidade do planejamento de produção e das necessidades de modelagem da produção nos ambientes fabris. Assim, a divisão de lotes em tamanhos variáveis foi utilizada por possibilitar uma flexibilidade ao modelo e proporcionar uma solução de escalonamento mais aderente às necessidades da vida real.

Neste entendimento, é possível verificar que a exploração de outros itens poderão contribuir para que o modelo represente ainda mais fidedignamente a realidade do processo produtivo, tais como:

- **Necessidades de material**: através da lista de materiais necessários em cada operação, a solução deverá verificar se há quantidade suficiente em estoque dos componentes necessários na operação no momento do escalonamento do lote. Caso não haja material disponível, a operação deve ser postergada ou até mesmo não escalonada, gerando um alerta;
- **Lotes em execução**: a otimização do escalonamento deve ter a possibilidade de receber a informação de quais lotes estão atualmente em execução na fábrica, pois assim poderá avaliar se a melhor alternativa é manter o lote em execução na máquina atual ou alterar seu escalonamento, levando em conta as consequências como nova necessidade de setup e alteração nas datas de entrega;
- **Lotes fixados**: a otimização do escalonamento deve ter a possibilidade de receber a informação de quais lotes estão fixados para a execução na fábrica, pois assim poderá avaliar as melhores alternativas levando em consideração essa restrição de ter uma operação com escalonamento fixado em determinada máquina em determinado período;
- **Dependência entre ordens**: a solução deverá possibilitar a informação de dependência entre ordens de produção. Assim, uma ordem não poderá ser iniciada sem que sua predecessora tenha sido finalizada;
- **Número de recursos secundários variável**: a solução deverá possibilitar a utilização de um número variável de recursos secundários, pois em alguns casos, para um escalonamento correto, são necessários, além da máquina que é o recurso principal, também ferramentas, pessoas, etc.

Outro ponto a ser estudado em trabalhos futuros é a utilização de outras técnicas na implementação da busca local do AM. Um estudo aprofundado sobre o tema pode trazer bons resultados para o modelo proposto.

REFERÊNCIAS

ABDOUN, O.; ABOUCHABAKA, J.; TAJANI, C. **Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem**. International Journal of Emerging Sciences, vol. 2, p. 61-77, 2012.

AKHSHABI, M.; AKHSHABI, M.; KHALATBARI, J. **Solving flexible job-shop scheduling problem using clonal selection algorithm**. Indian Journal of Science and Technology, vol. 4, p. 1248-1251, 2011.

ASEFI, H.; JOLAI, F.; RABIEE, M.; ARAGHI, M. **A hybrid NSGA-II and VNS for solving a bi-objective no-wait flexible flowshop scheduling problem**. The International Journal of Advanced Manufacturing Technology, vol. 75, p. 1017–1033, 2014.

AZZI, A.; FACCIO, M.; PERSONA, A.; SGARBOSSA, F**. Lot splitting scheduling procedure for makespan reduction and machine capacity increase in a hybrid flow shop with batch production**. The International Journal of Advanced Manufacturing Technology, vol. 59, p. 775–786, 2012.

BAKER, K. **Lot streaming in the two-machine flow shop with setup times**. Annals of Operations Research, vol. 57, p. 1-11, 1995.

BANHARNSAKUN, A.; SIRINAOVAKUL, B.; ACHALAKUL, T. **Job Shop Scheduling with the Best-so-far ABC**. Engineering Applications of Artificial Intelligence, vol. 25, p. 583–593, 2012.

BARZEGAR, B.; MOTAMENI, H.; BOZORGI, H**. Solving Flexible Job-Shop Scheduling Problem Using Gravitational Search Algorithm and Colored Petri Net**. Journal of Applied Mathematics, vol. 2012, article id 651310, 2012.

BEHNAMIAN, J. **Particle swarm optimization-based algorithm for fuzzy parallel machine scheduling**. The International Journal of Advanced Manufacturing Technology, vol. 75, p. 883–895, 2014.

BEHNKE, D.; GEIGER, J. **Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers**. Helmut-Schmidt-Universität, Lehrstuhl für BWL, insbes. Logistik-Management, 2012.

BERKOUNE, D.; MESGHOUNI, K. **Resolution approach for multi-objective problems with uncertain demands**. European Journal of Operational Research, vol. 187, p. 403–414, 2008.

BOWMAN, E. **The schedule sequencing problem**. Operations Research vol. 7, p. 621-624, 1959.

BRUCKER, P.; SCHLIE, R**. Job-shop scheduling with multi-purpose machines**. Computing, vol. 45, p. 369-375, 1991.

BUSCHER, U.; SHEN, L**. An integrated tabu search algorithm for the lot streaming problem in job shops**. European Journal of Operational Research, vol. 199, p. 385–399, 2009.

BUSCHER, U.; SHEN, L. **An Integer Programming Formulation for the Lot Streaming Problem in a Job Shop Environment with Setups**. Proceedings of the International Multi Conference of Engineers and Computer Scientists, IMECS 2011, vol. II, Hong Kong, 2011.

BUSCHER, U.; SHEN, L**. Modelling the Lot Streaming Problem with Setup times in a Job Shop Manufacturing System**. Engineering Letters, vol. 19, p. 2-7, 2011.

CALIS, B.; BULKAN, S. **A research survey: review of AI solution strategies of job shop scheduling problem**. Journal of Intelligent Manufacturing, vol. 26, p. 961–973, 2015.

CANTÚ-PAZ, E. **Implementing Fast and Flexible Parallel Genetic Algoritms**. Handbook of Pratical Genetic Algoritms, 1998.

CAPES. **Periódicos Capes – Busca de periódicos**. Web page: http://www.periodicos.capes.gov.br/. Acessado em 01/06/2016.

ÇETINKAYA, F.; DUMAN, M. **Lot streaming in a two-machine mixed shop**. The International Journal of Advanced Manufacturing Technology. vol. 49, p. 1161–1173, 2010.

ÇETINKAYA, F. **Unit sized transfer batch scheduling in an automated two-machine flow-line cell with one transport agent**. The International Journal of Advanced Manufacturing Technology, vol. 29, p. 178–183, 2006.

CHAN, F.; WONGB, T.; CHAN, L. **An evolutionary algorithm for assembly job shop with part sharing**. Computers & Industrial Engineering, vol. 57, p. 641–651, 2009.

CHAN, F.; WONG, T.; CHAN, P. **Equal Size Lot Streaming to Job-shop Scheduling Problem Using Genetic Algorithms**. Proceedings of the 2004 IEEE International Symposium on Intelligent Control, Taipei, Taiwan, 2004.

CHAN, F.; WONG, T.; CHAN, L. **Lot streaming for product assembly in job shop environment**. Robotics and Computer Integrated Manufacturing, vol. 24, p. 321–331, 2008.

CHAN, F.; WONG, T.; CHAN, P. **Lot Streaming Technique in Job-shop Environment**. Proceedings of the 13th Mediterranean Conference on Control and Automation Limassol, Chipre, 2005.

CHANG, Y.; JENG, M. **A Neural Network Model for The Job-Shop Scheduling Problem With The Consideration of Lot Sizes**. Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, p. 202-207, 1995.

CHAKARAVARTHY, G.; MARIMUTHU, S.; SAIT, A. **Performance evaluation of proposed Differential Evolution and Particle Swarm Optimization algorithms for scheduling m-machine flow shops with lot streaming**. The International Journal of Advanced Manufacturing Technology, vol. 24, p. 175–191, 2013.

CHAMBERS, B.; BARNES, W. **Flexible Job Shop Scheduling by Tabu Search**. The University of Texas, Technical Report Series ORP96-09, Graduate Program in Operations Research and Industrial Engineering, 1996.

CHEN, J.; STEINER, G**. Lot streaming with attached setups in three-machine flow shops**. IIE Transactions, vol 30, p. 1075-1084, 1998.

CHEN, J.; STEINER, G. **Lot streaming with detached setups in three-machine flow shops**. European Journal of Operational Research, vol. 96, p. 591-611, 1996.

CHEN, J.; STEINER, G. **Approximation methods for discrete lot streaming in flow shops**. Operations Research Letters, vol. 21, p. 139-145, 1997.

CHEN, J.; STEINER, G. **On discrete lot streaming in no-wait flow shops**. IEE Transactions, vol 35, p. 91-101, 2003.

CHENG, M.; SARIN, S.; Singh, S. **Two-stage, single-lot, lot streaming problem for a 1 + 2 hybrid flow shop**. Journal of Global Optimization, 2015.

CHIU, H.; CHANG, J. **Cost models for lot streaming in a multistage flow shop**. Omega, vol. 33, 435-450, 2005.

CHO, D.; LEE, Y.; LEE, T.; GEN M. **An adaptive genetic algorithm for the time dependent inventory routing problem**. Journal of Intelligent Manufacturing, vol. 25, p. 1025–1042, 2014.

CUI, Z.; GU, X**. An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems**. Neurocomputing, vol. 148, p. 248–259, 2015.

DAWKINS, R. **The selfish gene**. Best Books, 1976.

DEFERSHA, F. **A comprehensive mathematical model for hybrid flexible flowshop lot streaming problem**. International Journal of Industrial Engineering Computations, vol. 2, p. 83–294, 2011.

DEFERSHA, F. **A simulated annealing with multiple-search paths and parallel computation for a comprehensive flowshop scheduling problem**. International Transactions in Operational Research, vol. 22, p. 669–691, 2015.

DEFERSHA, F.; CHEN, M. **Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem**. The International Journal of Advanced Manufacturing Technology, vol. 62, p. 249–265, 2012.

DOUSTHAGHIA, S.; MOGHADDAMB, R. **An economic lot and delivery scheduling problem with the fuzzy shelf life in a flexible job shop with unrelated parallel machines**. International Journal of Industrial Engineering Computations, vol. 3, p. 663–680, 2012.

DOUSTHAGHI, S.; MOGHADDAM, R.; Makui, A. **Solving the economic lot and delivery scheduling problem in a flexible job shop with unrelated parallel machines and a shelf life by a proposed hybrid PSO**. The International Journal of Advanced Manufacturing Technology, vol. 68, p. 1401–1416, 2013.

DRISS, I.; MOUSS, K.; LAGGOUN, A. **A new genetic algorithm for flexible job-shop scheduling problems**. Journal of Mechanical Science and Technology, vol. 29, p. 1273-1281, 2015.

EDIS, R.; ORNEK, A**. Simulation analysis of lot streaming in job shops with transportation queue disciplines**. Simulation Modelling Practice and Theory vol. 17, p. 442–453, 2009.

ELHAFSI, M. **An operational decision model for lead-time and price quotation in congested manufacturing systems**. European Journal of Operational Research, vol. 126, p. 355-370, 2000.

GODINHO FILHO, M.; BARCO, C.; TAVARES NETO, R. **Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (FMS): a literature survey, classification and analisys**. Flexible Services and Manufacturing Journal, vol. 26, p. 408-431, 2014.

FISHER, H.; THOMPSON, L**. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules**. Industrial Scheduling, pages 225–251, 1963.

GANAPATHY, V.; MARIMUTHU, S.; PONNAMBALAM, S. **Tabu Search And Simulated Annealing Algorithms for Lot-Streaming in Two-Machine Flowshop**. IEEE International Conference on Systems Man and Cybernetics, p. 4221-4225, 2004.

GANTT, H. **Work, Wages and Profit**. The Engineering Magazine, 1913.

GAREY, M.; JOHNSON, D.; STOCKMEYER, L. **Proceedings of the sixth annual ACM symposium on Theory of computing**, p. 47-63, 1974.

GAREY, M. **The Complexity of Flowshop and Jobshop**. Mathematics of Operations Research, vol. 1, p. 117-129, 1976.

GEIGER, M. **Genetic Algorithms for multiple objective vehicle routing**. Proceedings of the Metaheuristics International Conference, Portugal, p. 349-353, 2008.

GEIRSSON, E. **Rollout Algorithms for Job-Shop Scheduling**. M.Sc. thesis, Faculty of Industrial Engineering, University of Iceland, 2012.

GLOVER, F. **Tabu Search Part I**. Orsa, Journal of Computing, vol. 1, p. 191-207, 1989.

GOLDBERG, D.; LINGLE, R. **Alleles, loci, and the traveling salesman problem**. Proceedings of an International Conference on Genetic Algorithms and Their Applications, vol. 54, p. 154-159, 1985.

GOLDBERG, D. **Genetic algorithms in search, optimization and machine learning**. Adisson-Wesley, 1989.

GÓMEZ, P.; ANDRÉS, R.; ROMANO, C.; **A review of lot streaming in a flow shop environment with makespan criteria**. Journal of Industrial Engineering and Management, vol. 6, p. 761-770, 2013.

GRAHAM, L.; LAWLER, E.; LENSTRA, K.; KAN, G. **Optimization and approximation in deterministic sequencing and scheduling: a survey**. Annals of discrete mathematics, Vol. 5, p. 287-326, 1979.

HABCHI, G.; LABRUNE, C. **Study of lot sizes on job shop systems performance using simulation**. Simulation Practice and Theory, vol. 2, 277-289, 1995.

HOLLAND, J. H. **Adaptation in natural and artificial systems**. University of Michigan Press, Ann Arbor, Michigan, 1975.

HONGYAN, S.; LIANG, G.; QUANKE, P. **Discrete Artificial Bee Colony Algorithm for Lot-streaming Flowshop with Total Flowtime Minimization**. Chinese Journal of Mechanical Engineering, vol. 25, p. 990-1000, 2012.

HORNG, H.; HU, C.; CHENG T. **Evaluating the effectiveness of FDM in identifying important factors in adynamic flowshop**. Robotics and Computer Integrated Manufacturing, vol. 25, p. 894–900, 2009.

HOSSEINABADI, A.; SIAR, H.; SHAMSHIRBAND, S.; SHOJAFAR, M.; NASIR, M. **Using the gravitational emulation local search algorithm to solve the multi-objective flexible dynamic job shop scheduling problem in Small and Medium Enterprises**. Annals of Operations Research, vol. 229, p. 451–474, 2015.

HUANG R. **Multi-objective job-shop scheduling with lot-splitting production**. International Journal of Production Economics, vol. 124, p. 206–213, 2010.

HUANG, R.; YANG, C. **Overlapping production scheduling planning with multiple objectives - An ant colony approach**. International Journal of Production Economics, vol. 115, p. 163-170, 2008.

JEONG, H; WOO, S; KANG , S; PARK, J. **A Batch Splitting Heuristic for Dynamic Job Shop Scheduling Problem**. Computers & Industrial Engineering, vol. 33, p. 781-784, 1997.

KALIR, A.; SARIN, S. **Optimal Solutions for the Single Batch, Flow Shop, Lot-streaming Problem with Equal Sublots**. Decision Sciences, vol. 32, p. 387-398, 2001.

KALIR, A.; SARIN, S. **A near-optimal heuristic for the sequencing problem in multiple-batch flow-shops with small equal sublots**. Omega, vol. 29, p. 577-584, 2001.

KALIR, A.; SARIN, S. **Constructing Near Optimal Schedules for the Flow-Shop Lot Streaming Problem with Sublot-Attached Setups**. Journal of Combinatorial Optimization, vol. 7, p. 23–44, 2003.

KALIR, A.; SARIN, S. **Evaluation of the potential benefits of lot streaming in flow-shop systems**. International Journal of Production Economics, vol. 66, p. 131-142, 2000.

KARABOGA D.; GORKEMLI, B.; OZTURK, C.; KARABOGA, N. **A comprehensive survey: artificial bee colony (ABC) algorithm and applications**. Artificial Intelligence Review, vol. 42, p. 21–57, 2014.

KENYON, G.; CANEL, C.; NEUREUTHER, B. **The impact of lot-sizing on net profits and cycle times in the n-job, m-machine job shop with both discrete and batch processing**. International Journal of Production Economics, vol. 97, p. 263–278, 2005.

KIM, K.; JEONG, I**. Flow shop scheduling with no-wait flexible lot streaming using an adaptive genetic algorithm**. The International Journal of Advanced Manufacturing Technology, vol. 44, p. 1181–1190, 2009.

KIS, T. **Job shop scheduling problem with processing alternatives**. Advanced Logistic Systems, 2014.

KHALILI, M.; NADERI, B. **A bi-objective imperialist competitive algorithm for no-wait flexible flow lines with sequence dependent setup times**. The International Journal of Advanced Manufacturing Technology, vol. 76, p. 461–469, 2015.

KOIKE , M.; TSUMAYA, A.; MATSUI, R.; MORINAGA, E.; WAKAMATSU, H.; ARAI, E. **Production scheduling system with dynamic lot size in case of considering set-up time**. Manufacturing Systems and Technologies for the New Frontier, Springer, p. 289-292, 2008.

KUMAR, V.; PANDEY, M.; TIWARI, M.; BEN-ARIEH D. **Simultaneous optimization of parts and operations sequences in SSMS: a chaos embedded Taguchi particle swarm optimization approach**. Journal of Intelligent Manufacturing, vol. 21, p. 335–353, 2010.

LI, J.; PAN, Q**. Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm**. Information Sciences, vol. 316, p. 487–502, 2015.

LIU, C. **Scheduling Jobs with Values Exponentially Deteriorating over Time in a Job Shop Environment**. Proceedings of the International Multi Conference of Engineers and Computer Scientists IMECS 2011, Hong Kon, 2011.

LIU, G.; LUH, P**. Scheduling Job Shops with Transfer Lots**. Proceeding of the 1996 IEEE Intemational Conference on Robotics and Automation, Minneapolis, Minnesota, 1996.

LIU, J. **Single-job lot streaming in m-1 two-stage hybrid flowshops**. European Journal of Operational Research, vol. 187, p. 1171–1183, 2008.

LIU, S. **A heuristic method for discrete lot streaming with variable sublots in a flow shop**. The International Journal of Advanced Manufacturing Technology, vol. 22, p. 662–668, 2003.

LOW, C.; HSU, C.; HUANG, K.; **Benefits of lot splitting in job-shop scheduling**. The International Journal of Advanced Manufacturing Technology, vol. 24, p. 773–780, 2004.

LUSTOSA, L. **Planejamento e Controle de Produção**. Elsevier, 2008.

MARIMUTHU, S.; PONNAMBALAM, S.; JAWAHAR, N. **Evolutionary algorithms for scheduling m-machine flow shop with lot streaming**. Robotics and Computer Integrated Manufacturing, vol. 24, p. 125–139, 2008.

MARIMUTHU, S.; PONNAMBALA, S. **Heuristic search algorithms for lot streaming in a two-machine flowshop**. The International Journal of Advanced Manufacturing Technology, vol.27, p. 174–180, 2005.

MARTIN, C. **A hybrid genetic algorithm/mathematical programming approach to the multi-family flowshop scheduling problem with lot streaming**. Omega, vol. 37, p. 126–137, 2009.

MOHTASHAMI, A. **A new hybrid method for buffer sizing and machine allocation in unreliable production and assembly lines with general distribution time-dependent parameters**. The International Journal of Advanced Manufacturing Technology, vol. 74, p. 1577–1593, 2014.

MOKHTARI, H. **A two-stage no-wait job shop scheduling problem by using a neuro-evolutionary variable neighborhood search**. The International Journal of Advanced Manufacturing Technology, vol. 74, p. 1595–1610, 2014.

MORTEZAEI, N.; ZULKIFLI, N. **A Study on Integration of Lot Sizing and Flow Shop Lot Streaming Problems**. Arabian Journal for Science and Engineering, vol. 39, p. 9283–9300, 2014.

MOSCATO, P.; NORMAN, M. **A "memetic" approach for the travelling sales-man problem - implementation of a computational ecology for combinatorial optimization on message-passing systems**. Proceedings of the International Conference on Parallel Computing and Transputer Applications, IOS Press, Amsterdam, 1992.

NASAB, M.; MODARRES M.; SEYEDHOSEINICA M. **A self-adaptive PSO for joint lot sizing and job shop scheduling with compressible process times**. Applied Soft Computing, vol. 27, p. 137–147, 2015.

NASAB, M.; SEYEDHOSEINI, S. **Multi-level lot sizing and job shop scheduling with compressible process times: A cutting plane approach**. European Journal of Operational Research, vol. 231, p. 598–616, 2013.

NEJATI, M.; MAHDAVI, I.; HASSANZADEH, R.; AMIRI, N.; Mojarad, M. **Multi-job lot streaming to minimize the weighted completion time in a hybrid flow shop scheduling problem with work shift constraint**. The International Journal of Advanced Manufacturing Technology, vol. 70, p. 501–514, 2014.

NERI, F. **Handbook of memetic algorithms**. Springer, 2012.

NOWICKI, E.; SMUTNICKI, C. **A fast tabu search algorithm for the permutation flow-shop problem**. European Journal of Operational Research, vol. 91, p. 160-175, 1996.

OUENNICHE, J.; BOCTOR, F. **The two-group heuristic to solve the multi-product, economic lot sizing and scheduling problem in flow shops**. European Journal of Operational Research, vol. 129, p. 539-554, 2001.

OZCAN, U.; TOKLU, B. **A new hybrid improvement heuristic approach to simple straight and U-type assembly line balancing problems**. Journal of Intelligent Manufacturing, vol. 20, p. 123–136, 2009.

PAN, Q.; TASGETIREN, M.; SUGANTHAN, P.; CHUA, T. **A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem**. Information Sciences, vol. 181, p. 2455–2468, 2011.

PAN, Q.; SUGANTHAN, P.; LIANG, J.; TASGETIREN, M. **A local-best harmony search algorithm with dynamic sub-harmony memories for lot-streaming flow shop scheduling problem**. Expert Systems with Applications, vol. 38, p. 3252–3259, 2011.

PAN, Q.; WANG, L.; GAO, L.; LI, J. **An effective shuffled frog-leaping algorithm for lot-streaming flow shop scheduling problem**. The International Journal of Advanced Manufacturing Technology, vol. 52, p. 699–713, 2011.

PAN, Q.; RUIZ, R. **An estimation of distribution algorithm for lot-streaming flow shop problems with setup times**. Omega, vol. 40, p. 166–180, 2012.

PATALIA, T.; KULKARNI, G. **Behavioral Analysis of Genetic Algorithm for Function Optimization**. Computational Intelligence and Computing Research, IEEE International Conference, p. 1-5, 2010.

PENG, Z.; LING, W.; SHENG, W. **A Discrete Fruit Fly Optimization Algorithm for Flow Shop Scheduling Problem with Intermingling Equal Sublots**. Proceedings of the 33rd Chinese Control Conference, China, p. 7466-7471, 2014.

PENHA, R.; ANDRADE, D.; BERGMANN, D.; MESQUIRA, A**. Aplicação de modelos matemáticos para a resolução de job**. Anais do III Simpósio Internacional de Projetos, São Paulo, 2014.

PERES, S.; LASSERRE, J. **An iterative procedure for lot streaming in job-shop scheduling**. Computers and Industrial Engineering, vol. 25, p. 231-234, 1993.

PETROVIC, S.; FAYAD, C.; PETROVIC, D.; BURKE, E.; KENDALL, G.; **Fuzzy job shop scheduling with lot-sizing**. Annals of Operations Research, vol. 159, p. 275–292, 2008.

QING, R.; WANG, Y. **Inventory based bi-objective flow shop scheduling model and its hybrid genetic algorithm**. Mathematical Problems in Engineering, article id 976065, 2013.

RADCLIFFE, N.; SURRY, P**. Formal Memetic Algorithms**. Evolutionary Computing: AISB Workshop, Springer-Verlag, p. 1-16, 1994.

RAMASESH, R.; Fu, H; FONG, D.; HAYYA, J. **Lot streaming in multistage production systems**. International Journal of Production Economics, vol. 66, p. 199-211, 2000.

REEVES, C. **Handbook of Metaheuristics**. Kluwer, 2003.

ROHANINEJAD, M.; KHEIRKHAH, A.; FATTAHI, P. **Simultaneous lot-sizing and scheduling in flexible job shop problems**. The International Journal of Advanced Manufacturing Technology, vol. 78, p. 1–18, 2015.

ROOBEEK, F. **A better choice-a fuzzy logic based lot sequencing decision support system for operators in a job shop fab with reentrant process flows**. Proceedings of IEEE International Symposium on Semiconductor Manufacturing, p. 37-40, 1997.

ROSSITA, D.; TOHMÉC, F.; Frutosa, M.; Barde, J.; Brozd, D. **A non-permutation flowshop scheduling problem with lot streaming: A Mathematical model**. International Journal of Industrial Engineering Computations, vol. 7, p. 507–516, 2016.

ROUNDY, R.; CHEN, D.; CHEN, P.; CAKANYILDIRIM, M.; FREIMER, M.; MELKONIAN, V. **Capacity-driven acceptance of customer orders for a multi-stage batch manufacturing system**. IEE Transactions, vol. 37, p. 1093-1105, 2005.

SANGSAWANG, C.; SETHANAN, K.; FUJIMOTO, T.; GEN, M. **Metaheuristics optimization approaches for two-stage reentrant flexible flow shop with blocking constraint**. Expert Systems with Applications, vol. 42, p. 2395–2410, 2015.

SARIN, S.; KALIR, A.; CHEN, M. **A single-lot, unified cost-based flow shop lot-streaming problem**. International Journal of Production Economics, vol. 113, p. 413–424, 2008.

SARIN, S.; JAIPRAKASH, P. **Flow Shop Lot Streaming**. Springer Science & Business Media, p. 1-3, 2007.

SAWIK, T. **Integer Programming Approach to Production Scheduling for Make-To-Order Manufacturing**. Mathematical and Computer Modelling, vol. 41, p. 99-118, 2005.

SEN, A.; BENLI, O. **Lot streaming in open shops**. Operations Research Letters, vol. 23, p. 135-142, 1999.

SEN, A.; TOPALOGLU, E.; BENLI, O. **Optimal streaming of a single job in a two-stage flow shop**. European Journal of Operational Research, vol. 110, p. 42-62, 1998.

SHARMA, P.; JAIN, A. **Stochastic Dynamic Job Shop Scheduling with Sequence Dependent Setup Times: Simulation Experimentation**. Journal of Engineering and Technology, vol. 5, p. 19-25, 2015.

SIMON, Z. **Examination of Scheduling Methods for Production Systems**. Advanced Logistic Systems, vol. 8, p. 111–120, 2014.

TORABI, S.; KARIMI, B.; GHOMI, S. **The common cycle economic lot scheduling in flexible job shops: The finite horizon case**. International Journal of Production Economics, vol. 97, 52–65, 2005.

TOSCANI, L.; VELOSO, P. **Complexidade de Algoritmos**, Bookman, 2008.

TSENG, C.; LIAO, C. **A discrete particle swarm optimization for lot-streaming flowshop scheduling problem**. European Journal of Operational Research, vol. 191, p. 360–373, 2008.

VARGA, Z. **Examination of scheduling methods for production systems**. Advanced Logistic Systems, 2014.

VENTURA, J.; YOON, S. **A new genetic algorithm for lot-streaming flow shop scheduling with limited capacity buffers**. Journal of Intelligent Manufacturing, vol. 24, p. 1185–1196, 2013.

VIANA, G. **Metaheurísticas e programação paralela em otimização combinatória**. UFC Edições, 1998.

VICKSON, R. **Optimal lot streaming for multiple products in a two-machine flow shop**. European Journal of Operational Research, vol. 85, p. 556-575, 1995.

VIJAYCHAKARAVARTHY, G.; MARIMUTHU, S.; SAIT, A**. Comparison of Improved Sheep Flock Heredity Algorithm and Artificial Bee Colony Algorithm for lot Streaming in m-Machine Flow Shop Scheduling**. Arabian Journal for Science and Engineering, vol. 39, p. 4285–4300, 2014.

WAGNER, J. ; RAGATZ, L. **The impact of lot splitting on due date performanc**e. Journal of Operations Management, vol. 12, p. 13-25, 1994.

WANGA, H.; Wub, Z.; Rahnamayan, S.; Sun, H.; Liu, Y.; Pan, J. **Multi-strategy ensemble artificial bee colony algorithm**. Information Sciences, vol. 279, p. 587–603, 2014.

WEAVER, Patrick. **A Brief History of Scheduling**. Mosaic Project Services Pty Ltd, 2006.

WEEDA, P. **On the choice of batch mode in order to maximize throughput**. Engineering Costs and Production Economics, vol. 19, p. 24l-247, 1990.

WONGA, T.; CHAN, F.; CHAN, L. **A resource-constrained assembly job shop scheduling problem with Lot Streaming technique**. Computers & Industrial Engineering, vol. 57, p. 983–995, 2009.

WONG, T.; NGAN, S. **A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop**. Applied Soft Computing, vol. 13, p. 1391–1399, 2013.

XU, X.; LI, L.; FAN, L.; ZHANG, J.; YANG, X.; WANG, W. **Hybrid discrete differential evolution algorithm for lot splitting with capacity constrains in flexible job shop**. Mathematical Problems in Engineering, article id, 986218, 2013.

YALAOUI, F.; CHU, C. **An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times**. IIE Transactions, vol. 35, p. 183-190, 2003.

YOON, S.; VENTURA, J. **A genetic algorithm for lot-streaming**. Proceedings of IIE Annual Conference, 2002_a.

YOON, S.; VENTURA, J. **An application of genetic algorithms to lot-streaming flow shop scheduling**. IIE Transactions, vol. 34, p. 779–787, 2002_b.

YOON, S.; VENTURA, J. **Minimizing the mean weighted absolute deviation from due dates in lot-streaming flow shop scheduling**. Computers & Operations Research, vol. 29, p. 1301-1315, 2002_c.

YURTKURAN, A.; EMEL, E. **A Modified Artificial Bee Colony Algorithm for p-Center Problems**. The Scientific World Journal, article id 824196, 2014.

ZHANG, W.; YIN, C.; LIU, J.; LINN, R. **Multi-job lot streaming to minimize the mean completion time in m-1 hybrid flowshops**. International Journal of Production Economics, vol. 96, p. 189–200, 2005.

ZHAO, Y.; WANG, H.; XU, X.; WANG, W. **A new hybrid parallel algorithm for consistent-sized batch splitting job shop scheduling on alternative machines with forbidden intervals.** The International Journal of Advanced Manufacturing Technology, vol. 48, p. 1091–1105, 2009.

ZHAO, Y.; WANG, H.; XU, X.; WANG, W. **New Hybrid Parallel Algorithm for Variablesized Batch Splitting Scheduling with Alternative Machines in Job Shops**. Chinese Journal of Mechanical Engineering, vol. 23, p. 484-495, 2010.