

UNIVERSIDADE NOVE DE JULHO (UNINOVE)
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO
(PPGEP)

MARCOS FERNANDO MACHADO DE JESUS DE SANTANA

MÉTODOS DE BUSCA LOCAL EM PROBLEMAS DE ESCALONAMENTO DA
PRODUÇÃO EM AMBIENTES JOB SHOP

São Paulo

2017

MARCOS FERNANDO MACHADO DE JESUS DE SANTANA

**MÉTODOS DE BUSCA LOCAL EM PROBLEMAS DE ESCALONAMENTO DA
PRODUÇÃO EM AMBIENTES JOB SHOP**

Dissertação de Mestrado apresentado ao Programa de Pós-Graduação em Engenharia de Produção (PPGEP) da Universidade Nove de Julho - Uninove, como parte dos requisitos para obtenção do título de Mestre em Engenharia de Produção.

Prof. Fábio H. Pereira, Dr. - Orientador

São Paulo

2017

Santana, Marcos Fernando Machado de Jesus de.

Métodos de busca local em problemas de escalonamento da produção em ambientes job shop. / Marcos Fernando Machado de Jesus de Santana. 2017.

78 f.

Dissertação (mestrado) – Universidade Nove de Julho - UNINOVE, São Paulo, 2017.

Orientador (a): Prof. Dr. Fábio Henrique Pereira.

1. Escalonamento. 2. Job Shop Scheduling. 3. Metaheurísticos. 4. Algoritmo genético. 5. Abordagem indireta.

I. Pereira, Fábio Henrique. II. Título

CDU 658.5

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE DISSERTAÇÃO

DE

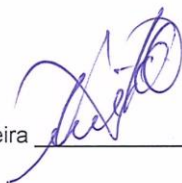
Marcos Fernando Machado de Jesus de Santana

Título da Dissertação: Métodos de Busca Local em Problemas de Escalonamento da Produção em Ambientes Job Shop.

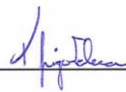
A Comissão Examinadora, Composta Pelos Professores Abaixo, Considero(a) o(a) candidato(a) Marcos Fernando Machado de Jesus de Santana Aprovado.

São Paulo, 30 de agosto de 2017.

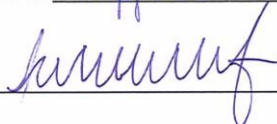
Prof(a). Dr(a). Fabio Henrique Pereira



Prof(a). Dr(a). Thiago Antonio Grandi de Tolosa



Prof(a). Dr(a). Sidnei Alves de Araújo



Dedico este trabalho a minha esposa, filha e pais por todo apoio, motivação, paciência e compreensão que foram dados, auxiliando-me na realização de mais uma grande conquista e sonho.

AGRADECIMENTOS

Deixo registrado neste trabalho meus agradecimentos, primeiramente à Deus pela capacidade e força que me foi dada para que fosse concluído com sucesso esta dissertação de mestrado em Engenharia de Produção.

Todos nós sabemos que toda conquista chega após uma determinada batalha e certamente essa batalha de ser reconhecido como Mestre em Engenharia de Produção pela Universidade Nove de Julho (UNINOVE) não foi fácil, compartilhar os estudos com a família, amigos e trabalho são tarefas complexas.

Quero agradecer a todos que contribuíram de forma direta e indireta para que essa batalha fosse vencida, porém em especial quero agradecer ao meu orientador professor Dr. Fábio Henrique Pereira, pois certamente sem as suas grandes contribuições, não teria chegado ao final deste trabalho. Professor muito dedicado, comprometido, amigo e um excelente orientador.

Agradeço também aos professores Doutores: Sidnei Alves de Araújo e Thiago Antonio Grandi de Tolosa pelas contribuições, pois foram de grande valor para aumentar a qualidade deste trabalho. Também agradeço a todos os professores do programa de Mestrado em Engenharia de Produção pela sábia forma de transmitir o conhecimento.

Agradeço à Universidade Nove de Julho (UNINOVE) por ter oferecido o curso de Mestrado em Engenharia de Produção com a mais alta qualidade, estrutura do programa, corpo Docente, Coordenadores, Diretores, Reitores. Excelência do módulo internacional, no qual proporcionou a plenitude do curso.

Agradeço também ao Luis Carlos dos Santos Junior e Fabio Knupp pela parceria na codificação para os experimentos, ao Luigi Tavolaro Santini, André Santos e todos os colegas de sala que são parte desta conquista.

Por fim agradeço minha esposa Jéssica e minha filha Luiza, pela paciência e minha ausência em muitos momentos nestes dois últimos anos, sem vocês não teria conseguido.

“Listen to the advice and accept instructions, and in the end you will be wise.”

Holy Bible Proverbs 19:20

Resumo

Resolver o problema de escalonamento da produção representa uma importante tarefa do planejamento e controle da produção. Esse problema consiste, resumidamente, em definir uma sequência de realização das operações de produção para cada um dos recursos (máquinas) disponíveis. Trata-se de um problema complexo, especialmente em ambientes de produção do tipo *job shop* nos quais cada *job* é definido como um conjunto único de tarefas que devem ser processadas em uma ordem pré-definida e diferente da dos demais *jobs*. Esses são os chamados *Job Shop Scheduling Problems* (JSSP). Para problemas menores, métodos exatos têm sido considerados os mais indicados por encontrarem a solução ótima em tempos computacionais aceitáveis. Já para problemas maiores, que crescem de forma não linear em relação ao número de *jobs* e máquinas, soluções heurísticas têm sido mais utilizadas em função do custo computacional, ainda que não garantam encontrar a solução ótima. Os métodos heurísticos e metaheurísticos têm ganhado destaque na literatura, como é o caso do Algoritmo Genético (AG) que é baseado na teoria da evolução das espécies. Entretanto, o algoritmo genético sem a aplicação de uma técnica de busca local, que é uma busca na vizinhança de uma solução com objetivo de refiná-la, não tem apresentado resultados satisfatórios para o problema abordado. O objetivo deste trabalho é comparar diferentes representações em um método de busca local, em conjunto com o AG, para o problema de escalonamento em ambiente *job shop*. Para os testes deste trabalho, método de busca local foi avaliado em instâncias conhecidas na literatura. Foram comparados métodos de busca com vizinhanças definidas a partir de representações diretas e indiretas da solução no Algoritmo Genético. Os resultados mostram que o método com abordagem indireta definida a partir de representações indiretas da solução são mais efetivos para os problemas testados, comparado com a abordagem direta, especialmente em relação ao custo computacional.

Palavras Chaves: Escalonamento, *Job Shop Scheduling*, metaheurísticos, algoritmo genético, abordagem indireta

ABSTRACT

Solving the problem of production scheduling is an important task in production planning and control. This problem consists, in short, to define a sequence of realization of the production operations for each of the resources (machines) available. This is a complex problem, especially in job shop-type production environments in which each job is defined as a single set of tasks that must be processed in a predefined order and different from that of other jobs. These are the Job Shop Scheduling Problems (JSSP). For smaller problems exact methods have been considered the most indicated because they find the optimal solution in acceptable computational times. For larger problems, which grow in a non-linear way in relation to the number of jobs and machines, heuristic solutions have been more used as a function of computational cost, although they do not guarantee to find the optimal solution. The heuristic and metaheuristic methods have gained prominence in the literature, as is the case of the Genetic Algorithm (GA), which is based on the theory of evolution of the species. However, the genetic algorithm without the application of a local search technique, which is a search in the vicinity of a solution to refine it, does not present satisfactory results for the problem specifically addressed. The objective of this work is to compare different representations in a local search method, together with GA, for the scheduling problem in the job shop environment. For the tests of this work, local search method was evaluated in instances known in the literature. Search methods with defined neighborhoods from direct and indirect representations of the solution in the Genetic Algorithm were compared. The results show that methods with indirect approach defined from indirect representations of the solution are more effective for the problems tested, compared to the direct approach, especially in relation to the computational cost.

Keywords: Scheduling, Job Shop Scheduling, metaheuristics, genetic algorithm, indirect approach

LISTA DE FIGURAS

Figura 1 - Ilustração de um problema de escalonamento da produção.	20
Figura 2 - Número de trabalhos publicados e disponíveis na base <i>sciencedirect</i> nos últimos cinco anos, buscando-se apenas pelo termo <i>scheduling</i>	21
Figura 3 - Exemplo de problema com uma única máquina	22
Figura 4 - Exemplo do ambiente de Máquinas idênticas em paralelo	23
Figura 5 - Exemplo de ambiente Flow Shop.	24
Figura 6 - Exemplo de ambiente Job shop.....	31
Figura 7 - Exemplo de um Diagrama de Gantt.	33
Figura 8 - Exemplo de uma forma de representação do tipo Gráficos Disjuntos.....	34
Figura 9 - Exemplo de representação por matrizes.....	35
Figura 10 - Representação do Alelo Cromossomo	37
Figura 11 - Estrutura Básica do Algoritmo Genético.....	38
Figura 12 - Princípio da roleta	39
Figura 13 - Cruzamento e Mutação	40
Figura 14 - Pseudocódigo do algoritmo do AG.	41
Figura 15 - Metodologia utilizando a representação binária.....	44
Figura 16 - Representação da matriz de prioridade	45
Figura 17 - Método de subida de encosta	50
Figura 18 - Elemento da solução antes e depois da permutação	51
Figura 19 - Elementos da solução de abordagem direta por inserção $S^* < S_{ws}$	52
Figura 20 - Elementos da solução de abordagem direta por de inserção $S^* > S_{wf}$	53
Figura 21 - Elementos da solução de abordagem direta por inserção completa	54
Figura 22 - Conceito de abordagem direta.....	55
Figura 23 - Conceito de abordagem indireta.....	56
Figura 24 - Fluxograma do AG com abordagem direta.....	58
Figura 25 - Fluxograma do AG com abordagem indireta.....	59
Figura 26 - Comparação dos melhores makespan obtidos com os resultados alcançados por abordagem indireta.	67
Figura 27 - Média de makespan alcançado comparado à outras representações	68
Figura 28 - Gráfico dos valores mínimos, médios e máximos de makespan obtidos	69

LISTA DE TABELAS

Tabela 1 - Exemplo de representação de sequência tecnológica.....	35
Tabela 2 - Parâmetros adotados no algoritmo genético	60
Tabela 3 - Comparação de makespan para as diferentes estratégias de busca.....	62
Tabela 4 - Comparação das melhores soluções obtidas neste trabalho com a literatura	66
Tabela 5 - Valores mínimos, médias e máximos de makespan obtido pela abordagem indireta.....	68

SUMÁRIO

1.	INTRODUÇÃO	14
1.1.	FORMULAÇÃO DO PROBLEMA	15
1.2.	OBJETIVOS	16
1.2.1.	Objetivo Geral	16
1.2.2.	Objetivos Específicos.....	16
1.3.	JUSTIFICATIVA.....	16
1.4.	ESTRUTURA DO TRABALHO.....	17
2.	FUNDAMENTAÇÃO TEÓRICA.....	19
2.1.	PROBLEMAS DE SCHEDULING	19
2.1.2.	Ambiente de produção.....	22
2.1.3.	Características de processamento e suas restrições	26
2.1.4.	Objetivo a ser minimizado.....	28
2.2.	JOB SHOP SCHEDULING PROBLEM.....	29
2.3.	REPRESENTAÇÃO DO PROBLEMA DE SCHEDULING	32
2.3.1.	Diagrama de Gantt.....	33
2.3.2.	Gráficos disjuntos	33
2.3.3.	Sequência Tecnológica	34
2.3.4.	Representação Matricial.....	35
2.4.	MÉTODOS DE SOLUÇÃO DO JSSP	36
2.5.	ALGORITMOS GENÉTICOS.....	37
2.5.1.	Codificação da solução no AG	41
2.5.2.	Representações Diretas	42
2.5.3.	Representações Indiretas	43
2.5.4.	Aplicação do AG em problemas de Scheduling	45
2.6.	BUSCA LOCAL.....	47
2.6.1.	Método de Subida/Descida de Encosta.....	49
2.6.2.	Definição do conceito de vizinhança de uma solução	50
2.6.3.	Abordagem direta por permutação	51
2.6.4.	Abordagem direta por inserção	52
2.6.5.	Abordagem direta por inserção completa.....	53
2.6.6.	Abordagem direta	54

2.6.7.	Abordagem indireta	55
3.	MÉTODOS E MATERIAIS	57
3.1.	MATERIAIS	61
4.	RESULTADOS E DISCUSSÕES.....	62
4.1.	CONFIGURAÇÃO DOS PARÂMETROS	62
4.2.	COMPARAÇÃO ENTRE AS ESTRATÉGIAS DE BUSCA LOCAL	62
5.	CONCLUSÃO	70
	REFERÊNCIAS	71

1. INTRODUÇÃO

O planejamento e controle da produção (PCP) é uma atividade corriqueira e muitas vezes problemática para as empresas de manufatura, nos mais diversos setores da indústria. Esse problema consiste em planejar e controlar o processo de fabricação definindo, entre outras coisas, quanto e quando produzir e quais recursos utilizar nessa produção (BORGES *et al.*, 2013). Portanto, uma das etapas do PCP consiste no escalonamento das ordens de produção nas máquinas no chão de fábrica, que consiste em definir a ordem em que cada máquina deverá processar as tarefas, com vistas a minimizar o tempo total de produção de um conjunto de itens.

Em geral, para executar uma ordem de produção diversas tarefas devem ser realizadas seguindo uma determinada rota nas máquinas. Esse conjunto de tarefas é chamado de *job* que é processado por todas as máquinas pertencentes à sua rota no processo produtivo. Nos casos em que as rotas, pré-definidas nas máquinas para cada um dos *Jobs*, são únicas e diferentes entre si, tem-se o chamado problema de sequenciamento da produção em *job shop*, ou *Job Shop Scheduling Problem* (JSSP), o qual é sempre mencionado como um dos problemas mais importantes e difíceis no contexto da programação da produção (ASADZADEH, 2015).

De fato, quanto à complexidade computacional o problema é classificado na literatura como NP-Hard, o que significa que a relação entre o tamanho do problema versus o tempo computacional para resolvê-lo cresce de forma não linear. A medida que o número de *jobs* e o número de máquinas crescem, o tempo computacional cresce exponencialmente o que faz com que o JSSP seja considerado como um membro de uma grande classe de problemas de difícil solução (NP-Hard), em especial com o uso de métodos exatos de otimização (JAIN, e MEERAN, 1998). Apesar dos avanços recentes e de garantirem encontrar a solução ótima, os métodos exatos são adequados apenas para problemas com um pequeno número de operações (*jobs* x máquinas).

Portanto, torna-se interessante a aplicação de técnicas heurísticas e metaheurísticas, que procuram encontrar uma solução de boa qualidade em um tempo computacional relativamente baixo, em comparação aos métodos exatos, em detrimento da otimalidade em muitos casos (MÓDOLO JUNIOR, 2015). Destaca-se

nesse contexto o os Algoritmos Genéticos (AGs) que, segundo Linden (2012), são técnicas metaheurísticas de otimização global que utilizam técnicas heurísticas para encontrar boas soluções dentro de um determinado espaço de busca.

Vale destacar, no entanto, que dada a complexidade do JSSP mesmo as mais sofisticadas metaheurísticas, e em especial o Algoritmo Genético, têm encontrado dificuldades para encontrar uma boa solução. Nessas circunstâncias, especialmente para problemas envolvendo muitos *jobs* e máquinas, torna-se necessário aplicar um método para refinar as soluções encontradas pela metaheurística. Esse método de refinamento de soluções é, em geral, um método de busca local, que é definido como alteração de uma solução por uma solução vizinha para melhorar a qualidade das soluções encontradas (ASADZADEH, 2015).

Um método de busca local (BL) guarda a solução atual e gradualmente tenta melhorá-la percorrendo a vizinhança dessa solução. Assim, além de uma função de avaliação de soluções, já disponível na metaheurística, devem ser definidos um conceito de vizinhança da solução e uma a estratégia de busca nessa vizinhança, ingredientes dos quais depende o sucesso da abordagem (FALCÃO, 2014). Consequentemente, a definição do método de busca local pode ser influenciada, em última análise, pela forma como a metaheurística representa as soluções do problema em questão.

1.1. FORMULAÇÃO DO PROBLEMA

Considerando que as soluções para o JSSP podem ser representadas de diferentes formas no contexto de uma metaheurística, a presente pesquisa busca responder a seguinte questão:

A forma de representação das soluções do JSSP no Algoritmo Genético e os diferentes conceitos de vizinhança definidos a partir dessas representações influenciam o desempenho da metaheurística?

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Aplicar métodos de busca local para o problema JSSP, em conjunto com o AG, com vistas a avaliar a influência de conceitos de vizinhanças definidos a partir de diferentes representações cromossômicas no AG para solução do JSSP.

1.2.2. Objetivos Específicos

Pesquisar na literatura trabalhos que abordam os principais métodos de busca local e que usam o AG em conjunto para otimizar o sequenciamento da produção em ambiente *job shop*.

Detalhar o conceito básico do AG, em especial a forma de representação da solução.

Definir o conceito de vizinhança para cada um dos métodos de busca local, busca por permutação, busca por inserção e busca por inserção completa através das representações binárias.

Definir as representações diretas e indiretas de vizinhança para busca local, permutação, inserção e inserção completa em problemas de sequenciamento de produção.

Analisar e comparar os resultados alcançados pelos métodos de buscas propostos com os resultados já conhecidos na literatura e a identificação da correta definição da vizinhança.

1.3. JUSTIFICATIVA

Estudos da literatura sobre a utilização de métodos de busca local para problemas de sequenciamento de produção no processo industrial tem se destacado,

pela eficiência em encontrar soluções melhores para problemas considerados complexos.

A busca local, que pode ser considerada como refinamento de soluções através de metaheurísticas, é fundamental para encontrar soluções melhores do que as já encontradas. Os métodos de busca local têm se firmado como a alternativa mais interessante, é importante pesquisar formas eficientes de fazer isso.

Este trabalho mostra as técnicas de busca local na vizinhança com uso do AG por meio das representações diretas e indiretas.

No caso se a vizinhança for considerada muito grande, o custo computacional aumenta consideravelmente e se a vizinhança for muito pequena, pode não resolver (ZHOU, 2013).

Após a busca local, a solução no AG deve ser atualizada em caso de melhora, e que isso também depende da representação.

1.4. ESTRUTURA DO TRABALHO

O trabalho está estruturado da seguinte forma:

O Capítulo 1 apresenta uma introdução ao problema de escalonamento de produção em ambiente *Job Shop*. Apresenta também a pergunta de pesquisa, objetivos, delimitação do tema e a justificativa relacionados ao trabalho.

O Capítulo 2 apresenta uma revisão da literatura do problema de escalonamento de produção. É apresentado o problema *Job Shop* e suas representações; a estrutura e a técnica do Algoritmo Genético, os operadores genéticos de cruzamento, seleção e mutação; a utilização dos métodos de Busca Local baseada no método de descida de encosta e, por fim, as formas de representações diretas e indiretas.

O Capítulo 3 é dedicado a metodologia do trabalho. Nesse capítulo é explicado como foi definida a estrutura de abordagem indireta, os métodos de Busca Local baseados na representação direta, e finalmente é detalhado o experimento e planejamento utilizado no problema *Job Shop*.

No Capítulo 4 são apresentados os resultados computacionais obtidos utilizando instâncias da literatura.

No Capítulo 5 são apresentadas as conclusões do trabalho e as perspectivas de pesquisas futuras.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados e discutidos os conceitos abordados nesta dissertação, a saber o problema de *scheduling* em ambientes *job shop*, o algoritmo genético e suas representações e os métodos de busca local. No contexto do *Job Shop Scheduling Problem* (JSSP), são apresentadas sua definição e formas de representação. Na sequência são discutidos os métodos de solução do problema, com destaque para as metaheurísticas e, em especial, o Algoritmo Genético. Por fim, é apresentado e discutido o conceito de métodos de busca local utilizados em conjunto com o Algoritmo Genético.

2.1. PROBLEMAS DE SCHEDULING

Os problemas de *scheduling* ou escalonamento da produção, são encontrados nos mais diversos setores de produção, em situações nas quais se tem a necessidade de alocar recursos limitados para realizar determinadas tarefas. Em ambientes de produção os recursos limitados são representados pelas máquinas e as tarefas são as operações realizadas nessas máquinas para produzir um conjunto de itens, conforme ilustrado na figura 1. Um job é definido como o conjunto de tarefas necessárias para a produção de um item, as quais são executadas pelas máquinas em uma ordem pré-estabelecida.

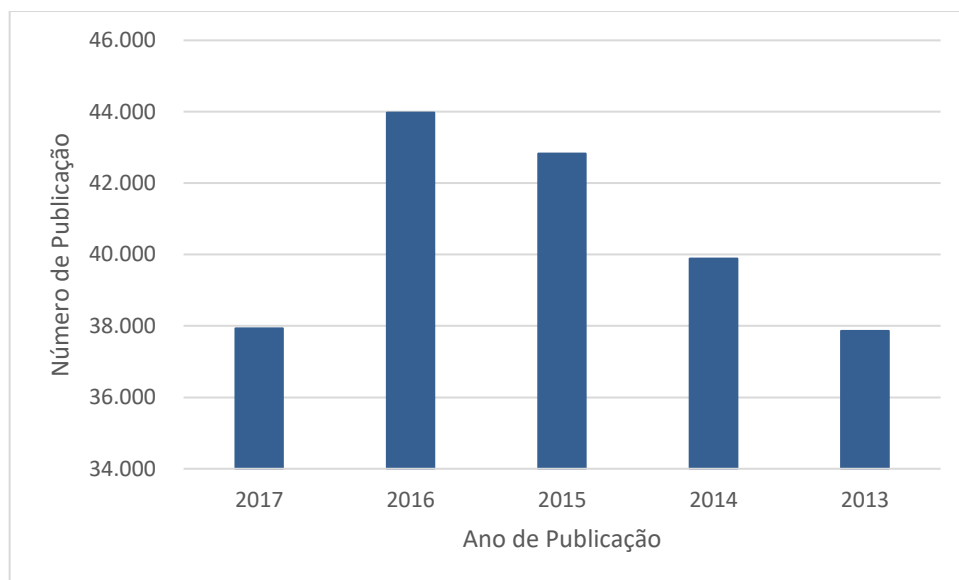
Figura 1 - Ilustração de um problema de escalonamento da produção.



Fonte: O autor

O problema de escalonar a produção, ou seja, determinar a ordem na qual essas tarefas devem ser processadas é, em geral, uma tarefa complexa, em especial quando o número de tarefas é grande (da ordem de centenas). Pode existir ainda a necessidade de levar em consideração diversas restrições do sistema, como espaço para armazenagem de matéria prima, local adequado para estoque, transferência de um local até outro para etapas diferentes de produção. Esses problemas possuem complexidade computacional elevada e, sendo assim, o estudo de abordagens de melhoria de processos e otimização do escalonamento da produção tornou-se um assunto de extrema relevância na literatura. Segundo Li *et al.* (2016) a produção *scheduling* tem uma das mais importantes regras do sistema de produção moderno de manufatura, além de contribuir para eficiência na manufatura, redução de conflitos, redução de *flow-time* e processo no trabalho. Para ilustrar essa relevância apresenta-se na figura 2 o número de trabalhos publicados, disponíveis na base *sciencedirect*, nos últimos 5 anos, buscando-se apenas pelo termo *scheduling* em pesquisa realizada em 2017.

Figura 2 - Número de trabalhos publicados e disponíveis na base *sciencedirect* nos últimos cinco anos, buscando-se apenas pelo termo *scheduling*



Fonte: O autor

2.1.1. Classificação dos problemas de *scheduling*

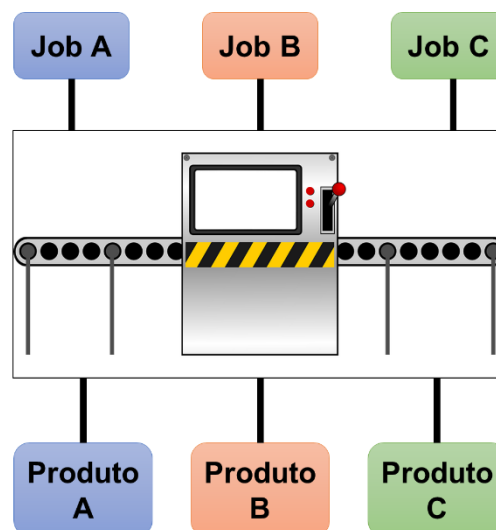
A classificação usual dos problemas de *scheduling* considera o tipo de ambiente de produção, a forma como o processamento é realizado e o objetivo a ser perseguido pelo escalonamento (PINEDO, 2008). A seguir é apresentada uma descrição detalhada dessa classificação, segundo notação proposta por Graham *et al.* (1979). Foram apresentados pelos autores as representações do ambiente de produção, e que contém apenas uma entrada, as características de processamento e suas restrições, e que pode não haver entradas, uma única entrada ou múltiplas entradas, o objetivo a ser minimizado e comumente contém apenas uma entrada. Foi utilizado o trabalho de Pinedo (2008) como referência para a classificação apresentada a seguir.

2.1.2. Ambiente de produção

A classificação em relação ao ambiente de produção estabelece as características do chão de fábrica quanto ao número e disposição das máquinas. Tem-se, nesse caso, uma divisão em ambiente de máquina única, máquinas idênticas em paralelo, máquinas diferentes em paralelo, ambientes *flow shop*, *flow shop* flexível, *job shop*, *job shop* flexível e *open shop* (PINEDO, 2008). Essa classificação é detalhada a seguir:

- ✓ Única Máquina (*Single machine*). Esse tipo de ambiente é o modelo mais simples e o mesmo serve como referência para se estudar heurísticas, que poderão ser aplicadas a modelos mais complexos. Para esse problema, todos os *jobs* e suas respectivas tarefas são processados em uma única máquina. Esse tipo de ambiente é ilustrado na Figura 3. Esse ambiente foi estudado por Biskup (1999), Kuo e Yang (2006), Gagné *et al.* (2002), Laguna *et al.* (1991), entre outros.

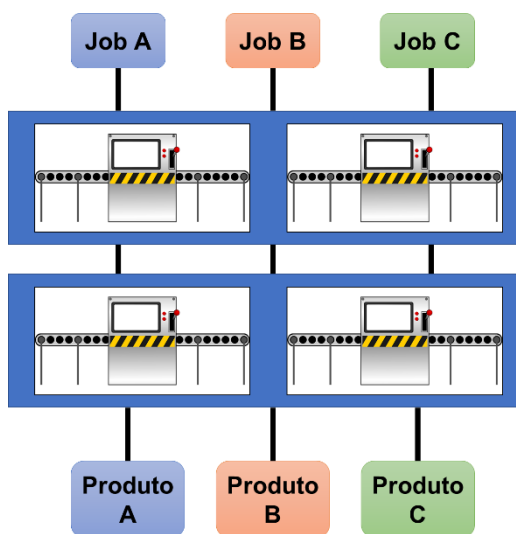
Figura 3 - Exemplo de problema com uma única máquina



Fonte: Autor

- ✓ Máquinas idênticas em paralelo (*Identical machines in parallel*). Nesse caso, o ambiente se dá na junção de máquinas idênticas de forma paralela formando um conjunto para executar o *job* que requer uma operação única e que pode ser processada por qualquer uma das máquinas do conjunto. Este ambiente é ilustrado na Figura 4. Esse ambiente foi estudado por Lee e Pinedo (1997), Hiraishi *et al.* (2002) entre outros.

Figura 4 - Exemplo do ambiente de Máquinas idênticas em paralelo

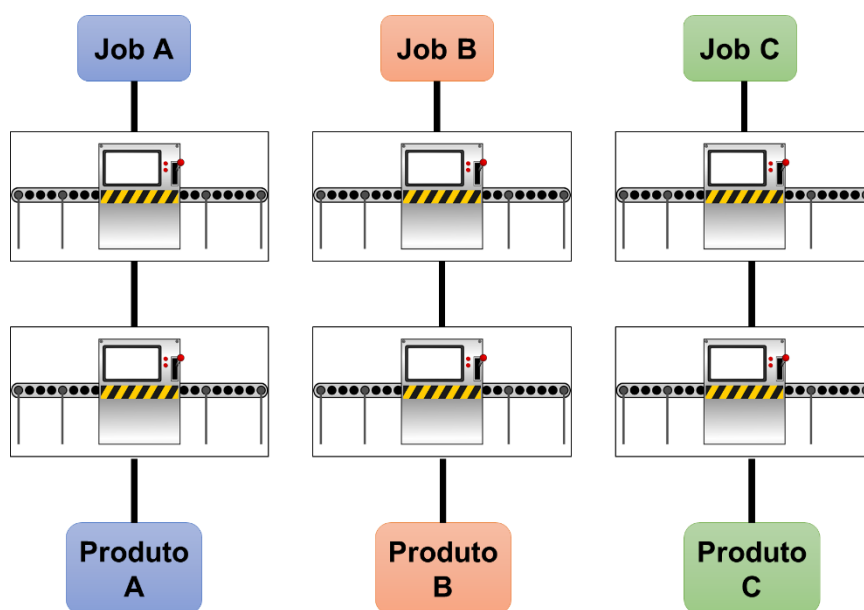


Fonte: Autor

- ✓ Máquinas em paralelo com diferentes velocidades (*Machines in parallel with different speeds*). Nesse ambiente, se assemelha com o ambiente anterior (Máquinas idênticas em paralelo), porém com restrições de tempo. Tem-se as mesmas máquinas em paralelo, só que diferentemente do cenário anterior que as máquinas tinham o mesmo tempo de execução da tarefa, nesse caso cada máquina tem um tempo diferente em que o tempo que o *job* gasta em uma máquina é dado pela razão tempo de processamento do *job* pela velocidade da máquina. Este ambiente também é conhecido como máquinas paralelas uniformes, e observa-se que o ambiente anterior é um subproblema desse ambiente. Como ilustrado na figura 4, o ambiente descrito é modificado apenas pelas diferentes velocidades das máquinas.

- ✓ Máquinas distintas em paralelo (*Unrelated machines in parallel*). Nesse ambiente, as máquinas distintas estão em paralelo, e as máquinas executam a mesma função, entretanto, seus recursos ou capacidades são diferentes. O objetivo do estudo nesse ambiente é a escolha da máquina mais apropriada para cada *job*. Nessa situação, o *job* será processado somente uma única vez em cada máquina; sendo que cada *job* possui um tempo de processamento no qual depende da máquina que será alocada.
- ✓ *Flow shop*. Para esse ambiente, existem m máquinas em série e cada *job* deve ser processado em cada uma das m máquinas, ou seja, o *job* deve passar por todas as máquinas seguindo um fluxo (*flow*) sequencial. Após a conclusão da tarefa em determinada máquina, aguarda-se em fila, para posteriormente ser processado e continuar a sequência. O processamento do *job* se dá pela regra FIFO (*first in, first out*), ou seja, o primeiro a entrar será o primeiro a sair, o *job* não pode seguir enquanto a máquina não terminar o processamento do *job*. A figura 5 ilustra um processamento em ambiente *Flow Shop*. Esse ambiente foi estudado por Ben-Daya e Al-Fawzan (1998), Yagmahan e Yenisey (2008), Ruiz e Vázquez-Rodríguez (2010) entre muitos outros, esse ambiente é relativamente bastante estudado na literatura.

Figura 5 - Exemplo de ambiente Flow Shop.



Fonte: Autor

- ✓ Flow Shop flexível. É uma generalização do problema *Flow shop* e máquinas em paralelo. Em vez de um número de máquinas em série, há estágios de processamento em que cada estágio tem-se um número de máquinas idênticas em paralelo. Por exemplo, o *job* tem que ser executado no estágio 1, depois no estágio 2 e assim por diante. Esse ambiente foi estudado por Wang (2005), Dai *et al.* (2013), entre outros.
- ✓ Job shop. Num ambiente do tipo *Job shop*, há um determinado número de máquinas em que cada *job* tem sua rota predeterminada para execução. Esse ambiente é o estudo do presente trabalho e será melhor apresentado na seção 2.1.3.
- ✓ Job shop flexível. Esse ambiente é uma generalização do *Job shop* com o ambiente de máquinas em paralelo. Nesse ambiente não há máquinas em série, ao contrário, tem-se centros de trabalhos com um número determinado de máquinas em paralelo. Os *jobs* tem que ser executados sequencialmente por esses centros e o *job* só deve ser executado em uma máquina do centro de trabalho e qualquer uma das máquinas pode executar o *job*. Esse ambiente foi estudado por Pezzella *et al.* (2008), Gao *et al.* (2008), Zhang *et al.* (2011), Gao *et al.* (2016), entre outros.
- ✓ Open shop. Em um ambiente Open shop, tem-se um número determinado de máquinas e cada *job* tem que ser processado novamente por cada uma das máquinas, porém alguns desses tempos de processamento pode ser zerado. Nesse ambiente, o programado pode determinar a rota para cada *job* e diferentes *jobs* podem ter diferentes roteiros. Esse ambiente foi estudado por Liaw (2000), Sha e Hsu (2008), Pongchairerks e Kachitvichyanuku (2016), entre outros.

2.1.3. Características de processamento e suas restrições

Neste contexto é classificado em fornecer detalhes de características e restrições de processamento e pode-se conter nenhuma entrada, uma única entrada ou várias entradas (PINEDO, 2008).

Essa classificação pode ser subdividida em: Data de lançamento (*Release dates*), Preempção (*Preemptions*), Restrições de precedência (*Precedence constraints*), Tempos de instalação dependente de sequência (*Sequence dependent setup times*), Famílias de jobs (*job families*), Processamento em lote (*Batch processing*), Avarias (*Breakdowns*), Restrições de elegibilidade da máquina (*Machine eligibility restrictions*), Permutação (*Permutation*), Bloqueio (*Blocking*), Sem espera (*No-wait*) e Recirculação (*Recirculation*).

- ✓ Data de lançamento (*Release dates*). A data de lançamento implica que, quando o *job* em um único problema de máquina tem datas de lançamentos diferentes, os problemas tendem a se tornar significativamente mais complicados. Quando a data de lançamento não é definida, o *job* pode ser processado em qualquer tempo.
- ✓ Preempção (*Preemptions*). A preempção, que significa uma precedência na compra, implica que o *job* não precisa necessariamente ser mantido na máquina, ou seja, o *job* com maior prioridade é processado imediatamente, ou com certa prioridade escolhida pelo programador, interrompendo o processamento do *job* com menor prioridade. Quando possível o *job* de menor prioridade volta a ser processado na máquina (ou outra máquina se elas estiverem em paralelo), continuando de onde parou ou podendo ser reiniciado. Isso é muito comum em processamento nas CPUs dos computadores.
- ✓ Restrições de precedência (*Precedence constraints*). Essas restrições de precedência comumente aparecem em ambientes de única máquina e máquinas paralelas, e requer que um ou mais *jobs* devem terminar antes de outro *job* estar autorizado a começar.
- ✓ Tempos de instalação dependente de sequência (*Sequence dependent setup times*). Nesse problema, existe uma determinada sequência para alguns *jobs* que precisam de uma nova configuração de máquina para continuar o processamento. Nessa configuração, tem-se um tempo de preparação da execução do *job j* e do

job k depende da máquina escolhida e também das características dos próprios *jobs* que serão processados e esse tempo é tido como restrição para o problema.

- ✓ Famílias de jobs (*job families*). Jobs da mesma família podem ter diferentes tempos de processamento, mas eles podem ser executados em uma mesma máquina sem que haja uma alteração na configuração, sendo assim interessante para otimizar o tempo total, pois não existirá o tempo de setup das máquinas.
- ✓ Processamento em lote (*Batch processing*). Nesse caso, uma máquina pode executar um número determinado de *jobs* simultaneamente. O processamento em lote comum em linhas de produção. O tempo de processamento dos *jobs* em lote podem ser o mesmo e só termina após o último *job* ser processado, isso implica em que o tempo total é determinado pelo *job* com maior tempo.
- ✓ Avarias (*Breakdowns*). Avarias da máquina condiz com a questão que as máquinas não estarão continuamente disponíveis. Essa indisponibilidade de uma ou mais máquinas é uma realidade nos problemas de escalonamento da produção e são considerados como uma restrição. Esse tempo em que as máquinas estão sendo arrumadas impacta na produção e sobrecarrega outras se o processamento for em paralelo. Dessa maneira, essa restrição é bastante considerada mesmo quando as manutenções são programadas.
- ✓ Restrições de elegibilidade da máquina (*Machine eligibility restrictions*). Esse tipo de restrição representa que não são todas as máquinas que tem a capacidade de processar um determinado *job* devido alguma restrição específica para o *job*.
- ✓ Permutação (*Permutation*). Em ambiente do tipo *Flow shop*, a ordem de escalonamento é única. Cada *job* deve passar em todas as máquinas que estão em série e, deve ser processado obrigatoriamente de acordo com a regra FIFO. Dessa maneira, a ordem ou permutação de cada *job* que passam pela primeira máquina é mantida em todo o sistema.
- ✓ Bloqueio (*Blocking*). Bloqueio normalmente acontece em ambiente do tipo *flow shop* e pode ser descrita como uma situação em que, o *flow shop* tem uma área de armazenamento limitado em duas máquinas sucessivas, isso representa que se essa área de armazenamento estiver cheia, as sequências das máquinas não poderão processar o *job* afetando toda a cadeia.
- ✓ Sem espera (*No-wait*). Esse é outro fenômeno que ocorre em ambiente do tipo *flow shop*, em que não permite que o *job* espere entre duas máquinas sucessivas,

ou seja, o tempo de processamento do *job* na primeira máquina deve ser calculado para que não haja espera para iniciar o processo na segunda máquina. Isso se dá para *jobs* que não podem esperar pois prejudicariam a integridade de determinado produto, exemplo que ocorre na linha da indústria metal/mecânica com produtos que devem operar em dadas temperaturas e não podem resfriar.

- ✓ Recirculação (*Recirculation*). Acontece em ambientes *job shop* e *low shop* quando um *job* pode passar por uma máquina ou por um centro de trabalho mais de uma vez.

2.1.4. Objetivo a ser minimizado

Em relação ao objetivo a ser minimizado, as seguintes medidas de desempenho podem ocorrer em problemas de escalonamento de tarefas de produção:

- ✓ Tempo de término (*Completion time* - C_{im}). Representa o tempo total de termino do *job* j na última máquina m .
- ✓ Tempo de fluxo (*flowtime* - F_j). Representa a soma dos tempos de espera e de processamento de um *job* j .
- ✓ Desvio do tempo de término (*Lateness* - L_j). Representa a diferença entre o tempo de término de um *job* e a data de entrega do *job*, se o tempo de entrega do *job* for maior que o término, o processo de entrega do *job* está atrasado, caso contrário significa que o processo de entrega do *job* está adiantado.
- ✓ Atrasados (*Tardiness* - T_j). Representa de forma semelhante ao parâmetro *lateness*, só que não considera os *jobs* que estiverem adiantados. Entende-se como o tempo de atraso dos *jobs* como $T_j = \max(L_j, 0)$.
- ✓ Antecipados (*Earliness* - E_j). Representa de forma semelhante ao *lateness*, considerando apenas os *jobs* que estiverem atrasados. Entende-se como o tempo de antecipação dos *jobs* como $E_j = \max(-L_j, 0)$.
- ✓ *Makespan* ($C_{máx}$). *Makespan* é o tempo total do começo ao fim da execução da operação, ou seja, representa o tempo total de conclusão do último *job* a sair do sistema. *Makespan* baixo significa uma boa utilização das máquinas.

- ✓ Tempo total de término ponderado (*total weighted completion time*). Busca minimizar a soma ponderada dos instantes de término de cada *job*, referentes ao peso atribuído.
- ✓ Desvio máximo em relação ao tempo do término dos *jobs* (*Maximum lateness*) ($L_{máx}$): Representa a pior violação das datas de vencimento.
- ✓ Soma ponderada dos atrasos (*Total weighted tardiness*). De forma análoga ao tempo total de término ponderado, essa função objetivo busca minimizar a soma ponderada dos instantes do atraso do *job*.
- ✓ Número total de *jobs* em atraso (*Number of tardy jobs*). Busca minimizar o maior número possível do atraso dos *jobs*.

Na classificação quanto ao ambiente de produção destaca-se neste trabalho o problema de *scheduling* em ambiente *job shop*, descrito detalhadamente a seguir.

2.2. JOB SHOP SCHEDULING PROBLEM

Segundo Asadzadeh (2015), o problema é descrito como um conjunto J_i de n *jobs* em que $i = 1, 2, \dots, n$ e esses *jobs* devem ser executados por conjunto M_k com uma quantidade m de máquinas disponíveis em que $k = 1, 2, \dots, m$. Dessa maneira, cada *job* representa uma cadeia de operações O , e os *jobs* devem ser executados em um determinado período de tempo ininterrupto em cada máquina. A operação do i -ésimo *job* e a k -ésima máquina é denotado como O_{ik} . Usualmente o problema é caracterizado por $n \times m$ em que n é o número de *jobs* e m o número de máquinas.

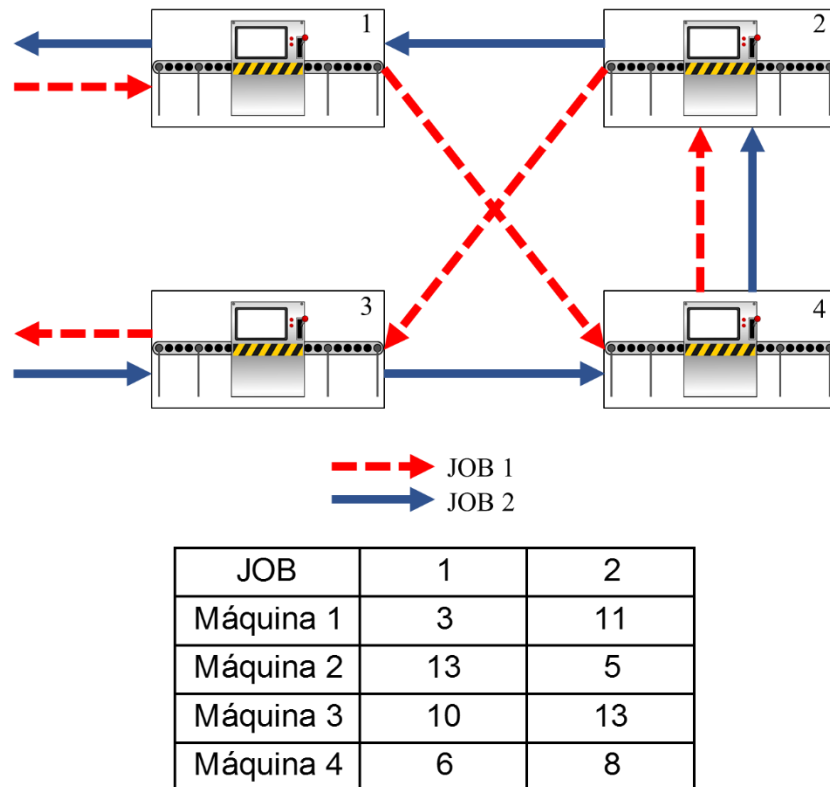
Segundo Fan e Zhang (2010) as características e restrições que diferenciam um *Job Shop Scheduling Problem* dos demais problemas de escalonamento, são definidas como segue:

- Todas as máquinas estão disponíveis no instante de tempo $t_0 = 0$ e todas as ordens são liberadas no instante $t_0 = 0$;
- Um *job* não pode ser executado duas vezes em uma mesma máquina;

- A sequência de visitação nas máquinas é chamada de sequência tecnológica e é fixa para cada *job* e é diferente de um *job* para outro;
- Não são permitidos processos simultâneos em qualquer uma das máquinas, o que significa que cada máquina somente executa um *job* por vez;
- Cada *job* é composto por várias tarefas a serem processadas;
- A sequência de tarefas para cada *job* é pré-definida e não pode ser alterada;
- Não é permitida preempção, ou seja, interrupção de processamento da tarefa;
- Não existem restrições de precedência nas operações de diferentes *jobs*.

Na figura 6 apresenta-se um exemplo de job shop em que existem $m = 4$ máquinas e $n = 2$ *jobs*. Na figura observam-se as sequências de tarefas para a execução dos *jobs* nas máquinas, ilustradas pelas setas tracejadas para o *job* 1 e contínuas para *job* 2. Assim, a sequência tecnológica é composta das máquinas M1, M4, M2 e M3 para o *job* 1 e M3, M4, M2 e M1 para o *job* 2. Indica-se também na Figura 6 um quadro com os tempos de processamento das tarefas para cada *job*. É apresentado no quadro as setas que representam a sequência tecnológica dos *jobs* e os tempos de processamento das tarefas de cada *job*.

Figura 6 - Exemplo de ambiente Job shop



Fonte: Autor

Podem existir ainda algumas restrições adicionais ao JSSP como por exemplo:

- ✓ *Distributed and Flexible Job shop Scheduling problem (DFJS)* - Considera o agendamento de ambientes de produção distribuídos, onde os *jobs* são processados por um sistema de várias Unidades de Manufatura Flexível (UMFs). Os problemas de agendamento distribuídos lidam com a atribuição dos jobs às UMFs e com a determinação do agendamento de cada UMF, em termos de atribuição de cada operação de trabalho a uma das máquinas capazes de trabalhar (flexibilidade de roteamento de tarefas) e sequência de operações em cada máquina.
- ✓ *Flexible job-shop scheduling problem (FJSP)* - O FJSP, que é uma extensão do JSSP clássico, permite que uma operação seja processada por qualquer máquina de um dado conjunto. O FJSP pode ser composto em dois sub-problemas: o

problema de seleção de máquina (MS) e o problema de escalonamento de operações (OS). O JSP contém apenas o problema de OS. Portanto, na mesma escala, o FJSP é mais complicado do que no JSP.

- ✓ *Job shop scheduling problem with total weighted tardiness objective (JSPTWT)* - Ambos os problemas diferem na função objetivo e nos requisitos adicionais do JSPTWT como permitir tempos de liberação de trabalho, considerando pesos de trabalho e mantendo datas de vencimento de trabalho. Devido à importância primordial do cumprimento das datas de vencimento dos trabalhos nas modernas cadeias de fornecimento, o JSPTWT tem recebido uma atenção substancial da prática e da academia nos últimos anos. Do ponto de vista prático, o atraso ponderado total dos empregos (TWT) observado para um fornecedor dentro de um período de referência determina a demanda ponderada total do cliente entregue no prazo.
- ✓ *Just-in-time job shop scheduling problem (JITJSSP)* - O trabalho deve ser concluído o mais próximo possível da data de vencimento. Uma conclusão precoce do trabalho resulta em custos de estocagem, como custos de armazenagem e seguro. Por outro lado, a conclusão do trabalho tardio resulta em penalidades, tais como perda de boa vontade do cliente e reputação danificada.

2.3. REPRESENTAÇÃO DO PROBLEMA DE SCHEDULING

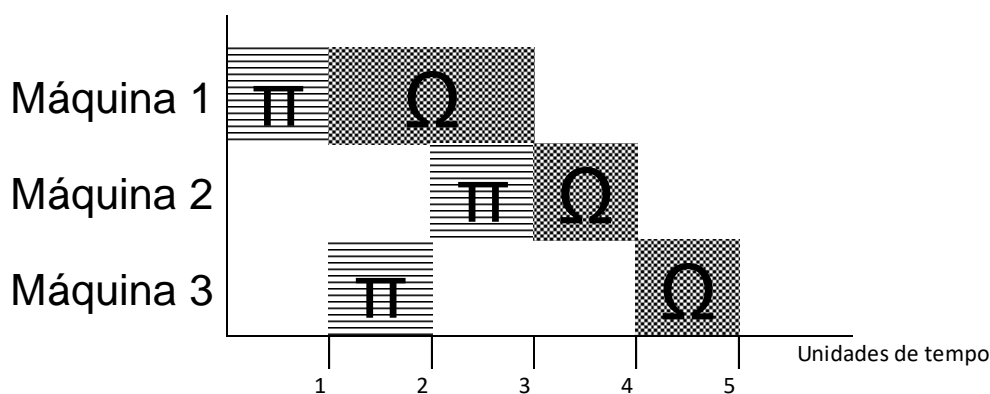
Existem diversas maneiras de representar os problemas de *scheduling*, com vistas a facilitar o entendimento e organização das informações e dos resultados dos problemas estudados. O problema *job shop* é algebricamente representado por $j/m/G/C_{m\acute{a}x}$, em que j representa a quantidade de *jobs*, m representa a quantidade de máquinas, G representa que o problema é do tipo *job shop* e $C_{m\acute{a}x}$ é a função objetivo de minimização do *makespan* (PINEDO, 2012).

Graficamente, o JSSP pode ser representado pelo diagrama de gantt e pelos gráficos disjuntos. Há ainda representações baseadas na sequência tecnológica e em matrizes, conforme apresentado nas próximas subseções.

2.3.1. Diagrama de Gantt

Essa forma de representação é bastante utilizada na literatura por ser bem sucinta e trazer todas as informações necessárias do problema. A figura 7 ilustra uma representação por diagrama de Gantt para um problema 2 X 3, sendo 2 *jobs* representados por π e Ω e 3 máquinas. Os blocos hachurados representam o tempo total que o *job* leva para ser processado, conforme eixo das abscissas dado em unidades de tempo.

Figura 7 - Exemplo de um Diagrama de Gantt.



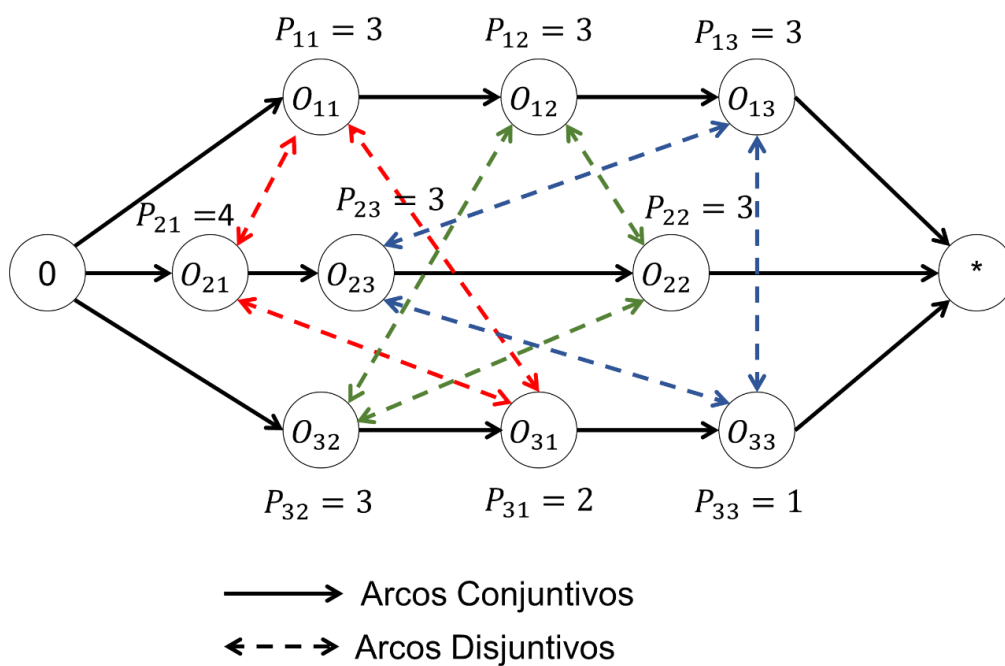
Fonte: Autor.

2.3.2. Gráficos disjuntos

Nesse tipo de representações, apresenta o roteiro do *job* de forma mais visual, facilitando a identificação da sequência em que o *job* é executado. Apresenta um grafo com as arestas direcionadas representando etapas da sequência tecnológica. As operações são representadas pelos vértices do grafo e definidas por O_{jm} que significa a operação do *job* j na máquina m . É também informado o tempo de processamento do *job* j na máquina m representado por P_{jm} . As orientações das setas tracejadas dos arcos disjuntos representam a prioridade da sequência de operações distintas na mesma máquina. Já as setas contínuas representam a ordem tecnológica de cada *job* tendo como origem o vértice zero e terminando no

vértice asterisco. A figura 8 ilustra uma representação de um gráfico disjunto para um JSSP com 3 *jobs* e 3 máquinas.

Figura 8 - Exemplo de uma forma de representação do tipo Gráficos Disjuntos



Fonte: Autor.

2.3.3. Sequência Tecnológica

Esta forma de representação é a mais intuitiva pois apresenta cada *job* com sua respectiva sequência de máquinas, bem como o tempo de processamento do *job*. A tabela 1 exemplifica a representação por sequência tecnológica.

Tabela 1 - Exemplo de representação de sequência tecnológica

<i>Job</i>	Máquina (Tempo de processamento)		
1	2(1)	3(1)	1(2)
2	3(2)	2(1)	1(1)
3	1(1)	2(1)	3(1)

Fonte: Autor.

2.3.4. Representação Matricial

Essa representação por matrizes de *jobs* e tempos de processamento é gerada a partir da representação de sequência tecnológica. A matriz representada pelo símbolo O_{jm} , apresenta a sequência de operações dos jobs nas máquinas e a matriz P_{jm} apresenta os respectivos tempos de processamento. A figura 9 ilustra como a representação por matrizes pode ser apresentada.

Figura 9 - Exemplo de representação por matrizes

$$\left\{ \begin{array}{l} O_{jm} = \begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 1 \\ 1 & 2 & 3 \end{bmatrix} \\ P_{jm} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \end{array} \right.$$

Fonte: Autor.

2.4. MÉTODOS DE SOLUÇÃO DO JSSP

Historicamente o JSSP foi inicialmente tratado pelas seguintes abordagens:

- ✓ Métodos exatos: Giffler e Thompson (1960), Brucker *et al.* (1994) e Williamson *et al.* (1997).
- ✓ *Branch and Bound*: Lageweg *et al.* (1977), Carlier e Pinson (1989), Applegate e Cook (1991), , Brucker *et al.* (1994) e Jain e Meeran (1998) e Sabuncuoglu e Bayiz (1999).

Porém, devido à dificuldade intrínseca do problema que faz com que métodos exatos se tornem computacionalmente inviáveis para problemas maiores, métodos heurísticos e metaheurísticos têm sido propostos para o problema, como nos trabalhos de Baker e French (1983), Gray e Hoesada (1991) e Gonçalves e Mendes (1994).

Métodos heurísticos são algoritmos que consideram algum conhecimento extra acerca do problema para guiar o processo de busca e tendem a encontrar uma solução próxima do ótimo. Uma heurística pode ser definida como uma técnica inspirada em processos intuitivos que procura uma solução factível, não necessariamente a melhor solução, para um determinado problema, com tempo computacional aceitável. Já as metaheurísticas se caracterizam por guiarem outras heurísticas e têm sido interessantes na resolução de problemas complexos de otimização (ARAÚJO *et al.*, 2010).

Umas das heurísticas que obtém melhores resultados para minimizar o *makespan* é o *Shifiting Bottleneck* do trabalho de Adams *et al.* (1988) e Asadzadeh (2015).

Atualmente os métodos metaheurísticos tem ganhado grande relevância na literatura para buscar a solução (sub-) ótima no problema de escalonamento de produção em ambiente *Job Shop*, sendo que, os métodos utilizados com maior destaque são: Busca Tabu (TS), *Simulated Annealing* (SA), *Variable Neighborhood Search* (método de busca de vizinhança variável - VNS), GRASP (*Greedy Randomized Adaptative Search Procedure*), Algoritmo Genético (AG), Colônia de Formigas e Nuvem de Partículas.

Em especial, os algoritmos genéticos, propostos por Holland (1975), têm sido utilizados com sucesso em uma variedade de problemas práticos. Davis (1985) aplicou primeiramente um algoritmo genético ao JSSP com sucesso, e atualmente os algoritmos genéticos tem provado serem capazes de encontrar boas soluções aproximadas para o JSSP.

2.5. ALGORITMOS GENÉTICOS

O algoritmo genético é baseado na teoria da evolução das espécies e na genética. É um método de busca e otimização baseado no processo de seleção natural que simula a evolução das espécies (GOLDBERG, 1989).

O algoritmo genético começa pela inicialização da população de cromossomos que representa um conjunto de soluções iniciais em que cada indivíduo da população representa uma solução inicial. Esses cromossomos podem ser codificados de diversas formas para representar de forma direta ou indiretamente as soluções do problema. Um exemplo de codificação binária é apresentado na Figura 10, com destaque para o alelo que é chamado assim, pois tem como base na genética os cromossomos que são formados por genes, que pode ter um determinado valor entre vários possíveis (LINDEN, 2012).

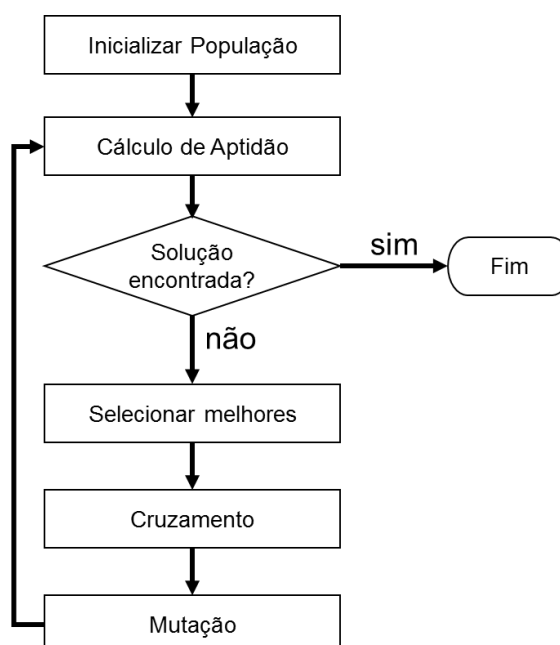
Figura 10 - Representação do Alelo Cromossomo

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

Fonte: O autor

A Figura 11 ilustra o fluxograma da estrutura básica do Algoritmo Genético, observa-se as etapas do algoritmo passando pelas seguintes etapas segundo (OMBUKI; VENTRESCA, 2004):

Figura 11 - Estrutura Básica do Algoritmo Genético



Fonte: o autor

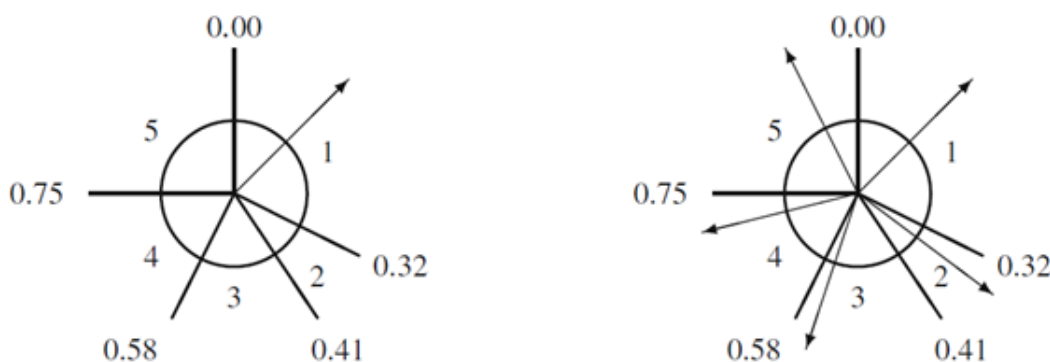
1. Inicialmente a estrutura básica do AG é composto por módulo de avaliação, módulo de população e o módulo de reprodução, no qual o módulo de avaliação é responsável pelo grau de adaptação de cada indivíduo (cromossomo), módulo de população é responsável por inicializar os cromossomos (*strings* de *bits*), criando a população inicial de indivíduos e módulo de reprodução que é responsável pela aplicação de operadores genético aos indivíduos (mutação e crossover) (SILVEIRA, 1998).
2. Realiza-se o cálculo de aptidão (fitness) de cada indivíduo da população, verifica-se qual indivíduo é mais apto a gerar novas soluções: No AG é necessário calcular a probabilidade de um indivíduo de sobreviver na próxima geração, esse cálculo depende do problema estudado.

3. Se não for encontrado a melhor solução, *seleciona os melhores indivíduos*.

O método mais comum para implementar a seleção dos indivíduos é o método da roleta (*roulette-wheel*). O princípio de seleção por roleta se baseia na evolução genética das espécies em que indivíduos mais bem adaptados tendem a sobreviver e repassar a genética para seus descendentes.

Pode-se então fazer uma alusão a uma roleta em um cassino. Usualmente, uma parte da roda de uma roleta é atribuído a cada uma das possíveis seleções com base no valor da aptidão. Dessa maneira pode-se alcançar, dividindo a aptidão de uma seleção pela aptidão total de todas as seleções. Para finalizar, escolhe-se de forma aleatória assim como é feito quando a roda da roleta é girada. Na figura 12 à esquerda, a roleta mostra os valores das probabilidades distribuídas pela roleta, à direita, após a atualização dessas prioridades conforme repetição na execução do algoritmo.

Figura 12 - Princípio da roleta



Fonte: (GENDREAU; POTVIN, 2010)

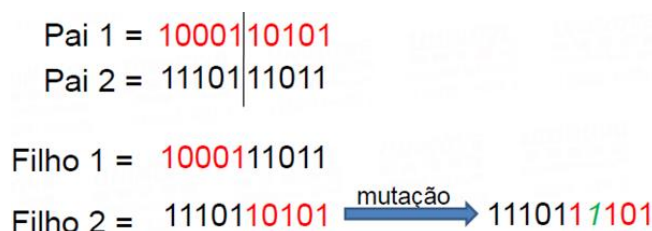
4. Selecionados os melhores indivíduos da população de cromossomos e realizado o cruzamento entre eles, também conhecido na literatura como crossover, isso promove a obtenção de melhores resultados preservando as características das melhores soluções conhecidas. Nesse contexto, este operador é o operador dominante pois através do cruzamento são criados novos indivíduos que preservam

as características boas de seus “pais”. Essa mistura tenta preservar as boas qualidades e tenta imitar de forma abstrata a reprodução dos genes nas células.

5. Aplica-se o processo de mutação. A Figura 13 ilustra como é gerada uma mutação binária no algoritmo genético. A mutação no cromossomo é aplicada para que a solução a ser encontrada saia dos mínimos locais. A mutação modifica de forma aleatória alguma característica do indivíduo, sendo muito importante para a introdução e manutenção da diversidade genética da população.

As operações de cruzamento e mutação são ilustradas na Figura 13. Os genes do Pai 1 se cruzam com os genes do Pai 2 formando os filhos que são as novas soluções do problema, observa-se que o filho 2 sofre uma mutação pois ajuda que a solução não fique estagnada em uma determinada posição e possibilita que se chegue em qualquer ponto do espaço de busca.

Figura 13 - Cruzamento e Mutação



Fonte: (ARAÚJO *et al.*, 2010)

Após a mutação retorna para o passo 2 até encontrar uma solução para o problema ou atingir o critério de parada geralmente definido por um número máximo de iterações, também chamadas gerações.

Em resumo, as características dos AGs são:

- ✓ Manipula uma população de indivíduos;
- ✓ Indivíduos/cromossomos são possíveis soluções de problemas;

- ✓ Os indivíduos são combinados (crossover) uns com os outros, podendo gerar herdeiros que podem sofrer ou não mutação.
- ✓ A evolução dos indivíduos e ou população se dá através de várias gerações até encontra soluções melhores, podendo ser quase ótima ou ótima.

A figura 14 mostra o pseudocódigo do algoritmo do AG. Nessa figura pode-se observar as etapas da figura 11.

Figura 14 - Pseudocódigo do algoritmo do AG.

```

Início
Seja  $s'$  a melhor solução da geração corrente,  $s^*$  a melhor solução obtida até então, Ger o contador do
número de gerações e Max_Ger o número máximo de gerações permitidas
Ger  $\leftarrow$  1
Gera população inicial
Avalia população {cálculo da aptidão dos indivíduos}
 $s^* \leftarrow \text{MelhorIndividuo(população inicial)}$ 
Enquanto Ger  $\leq$  Max_Ger ou algum outro critério de parada não for atingido
  Faz seleção dos melhores indivíduos
  Efetua o cruzamento entre os melhores indivíduos para recompor a população
  Faz mutação de um percentual dos indivíduos da população
  Avalia população {cálculo da aptidão dos indivíduos}
   $s' \leftarrow \text{MelhorIndividuo(população corrente)}$ 
  Se  $f(s') > f(s^*)$  então
     $s^* \leftarrow s'$ 
  Fim_se
  Ger  $\leftarrow$  Ger + 1
Fim_enquanto
Retorna  $s^*$ 
Fim

```

Fonte: Araújo *et al.* (2010)

2.5.1. Codificação da solução no AG

Os cromossomos que representam a solução do problema no AG podem ser codificados de diferentes formas. Além da codificação binária ilustrada na Figura 15,

podem ser usados, por exemplo, números inteiros e reais (SANTOS, *et al.*, 2016). Assim, a forma de codificação do cromossomo pode representar uma solução do problema de forma direta ou indireta. Se a solução do problema é um número decimal e a codificação é também decimal, de forma que não é necessária qualquer operação de conversão para transformar um cromossomo em uma solução, então diz-se que se trata de uma representação direta. Ao contrário, em representações indiretas é sempre necessária uma decodificação do cromossomo para obter uma solução do problema.

Para os problemas de escalonamento da produção, em especial, tem-se diversas formas de representação das soluções, tanto diretas quanto indiretas. Igualmente, nas representações diretas as possíveis soluções encontradas possuem ligação direta com o problema e podem ser avaliadas diretamente.

Já as representações indiretas necessitam do auxílio de um algoritmo de tratamento para que sejam geradas soluções, não sendo possível a análise do indivíduo gerado diretamente.

Exemplos de representação direta são a baseada em operações (*operation-based* - OB) e baseada em jobs (*job-based* – JB) e lista de preferência (*list-based* – PL). Para as representações indiretas destacam-se as representações baseadas em regras de despacho (*priority rule-based* – PR), baseada em gráfico disjuntivo, baseada em chaves aleatórias (*random key* – RK). Outro exemplo de representação indireta é a baseada em semente dinâmica propostas por Grassi (2014).

2.5.2. Representações Diretas

Representação OB: Para Gen *et al.* (1994) a representação baseada em operações, possui o tamanho de $j \times m$, no qual a letra j representa a quantidade de jobs e a letra m representa a quantidade de máquinas. Deve ser obedecida a regra tecnológica, ou seja, existe uma sequência para cada job em cada máquina pré-definidos. Essa representação gera um vetor, pois é realizada permutação dos jobs, para então ser aplicado o cruzamento.

Representação Job Based (JB): As representações baseadas em jobs são utilizadas para tratar problemas de escalonamento, no qual é necessário obedecer às

restrições tecnológicas. Nessa representação, os *jobs* tem que passar por todas as máquinas, não ao mesmo tempo porem deve ser aproveitado o tempo livre de cada máquina, considerando a sequência baseada no cromossomo do AG, em que j é o número de *jobs* e m o número de máquinas.

2.5.3. Representações Indiretas

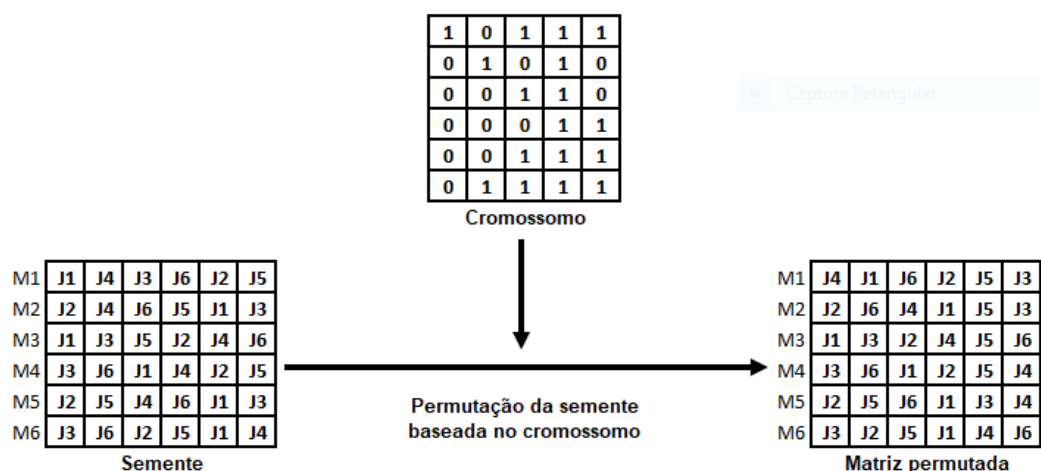
Representação *Priority rule-based* (PR) – Essa representação é baseada em regras de prioridade de despacho, é umas das representações mais utilizadas segundo Cheng (1996), considera a sequência conforme prioridade. Para a decodificação do cromossomo é utilizado o algoritmo de Giffler e Thompson (1960).

Representação *Random Key* (RK) – Semelhante a representação baseada em objeto (OB) descrito anteriormente, para problemas de escalonamento. Nesta representação os cromossomos são mostrados como *strings* de números reais, que são obtidos de maneira aleatória entre 0 e 1.

Representação por gráfico disjuntivo – Essa representação é orientada por arcos disjuntivos, no qual é um vetor binário com tamanho $j \times m$, não permitindo tarefas com execuções simultâneas nas mesmas máquinas.

Representação por semente dinâmica (SD) - nessa representação o cromossomo é definido como uma matriz binária bidimensional, conforme ilustrado na Figura 15.

Figura 15 - Metodologia utilizando a representação binária



Fonte: o autor

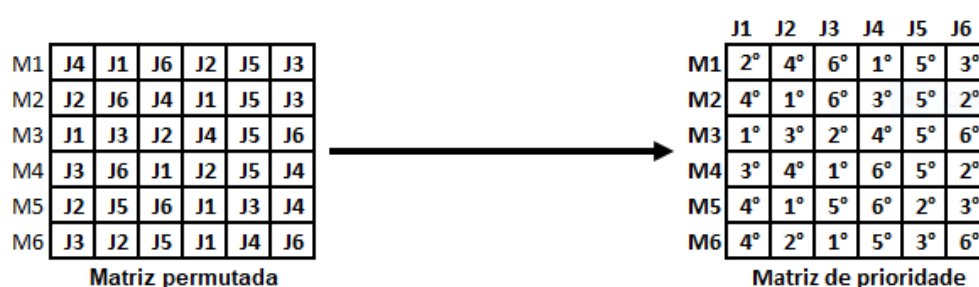
Conforme definido por Grassi (2014) foi proposto e avaliado uma nova forma de representação que tem por base uma matriz binária de tamanho $m \times (n-1)$. A representação que foi proposta não permite uma interpretação direta de uma possível solução. Desta forma foram desenvolvidos processos computacionais para geração de soluções a partir do cromossomo binário.

Inicialmente temos a matriz da semente que é a sequência de *jobs* por máquina, através da heurística *FIFO (First In, First Out)* que é a ordem de chegada do *job* e define-se a execução nas máquinas. É realizada uma leitura da matriz inicial por colunas, da esquerda para a direita, para cada máquina, em ordem crescente, como apresentado na Figura 15. Então observa-se que na máquina 1 chega primeiro o *job* 1, seguidos dos *jobs* 4, 3, 6, 2 e 5, que gera a primeira linha da 3ª matriz permutada após a aplicação do AG através da 2ª matriz binária. Repete-se o processo para a máquina 2, e assim sucessivamente, que gera a 3ª matriz permutada apresentada na Figura 15, que é a semente geradora das soluções. Essa transformação acontece uma única vez, no início do processo.

A matriz binária apresentada na Figura 15, representam os indivíduos do AG. A leitura da 2ª matriz cromossomo é realizada por linhas, de cima para baixo. Permuta-se na semente sempre que for identificado o número 1, essa permutação acontece na posição identificado com o número 1 com o seu sucessor. Nota-se que os *jobs* 1 (primeiro elemento) e 4 (segundo elemento) são permutados como primeiro elemento da nova matriz, permanecendo na posição 1 o *job* 4. Na sequência é verificado a

segunda posição do alelo no cromossomo, no caso é 0 que não deve ser alterado, permanecendo na mesma posição, que é o job 1. É observado que na 3ª matriz permutada os dois primeiros elementos são *jobs* 4 e 1. O processo de permutação entre os alelos do cromossomo é realizado para os alelos sequenciais da primeira linha e posteriormente da segunda linha, e assim por diante. Foi considerado para o problema FT06 apenas 5 colunas na matriz binária, pois é verificado a 6ª posição com relação à 5ª posição, não havendo necessidade de ter a 6ª coluna.

Figura 16 - Representação da matriz de prioridade



Fonte: o autor

A última matriz gerada através da matriz permutada é transformada em uma matriz de prioridade de *jobs* por máquina conforme Figura 16, no qual é estabelecido a ordem de execução em uma rotina desenvolvida em C, que na matriz permutada que foi o *job* 4 é executado primeiramente, na sequência o *job* 1 e pelo *job* 6. Então esses *jobs* na matriz de prioridades recebem as prioridades 1, 2 e 3, respectivamente. Realiza-se esse processo com todos elementos da matriz que podem gerar solução factíveis e não factíveis.

2.5.4. Aplicação do AG em problemas de Scheduling

Muitos trabalhos têm sido desenvolvidos objetivando aprimorar a eficiência do AG em problemas de *scheduling*. Por exemplo, um algoritmo genético com a adaptação da área de busca foi proposto por Someya e Yamamura (2001). Já no trabalho de Zhou *et al.* (2001) foi proposta uma formulação híbrida do GA para o JSSP, em que as regras da programação, tais como o menor tempo de processamento e o

maior número de trabalhos restantes, foram integrados no processo de evolução genética. No trabalho de Park *et al.* (2003), os autores desenvolveram um método eficiente baseado em algoritmo genético para abordar o JSSP, implementando um método de programação baseado em algoritmo genético paralelo. Já no trabalho de Mattfeld e Bierwirth (2004) consideraram um problema multiobjectivo da programação de trabalhos com liberação e datas devidas, bem como retardamento como objetivos. Watanabe *et al.* (2005) implementam um algoritmo genético modificado com uma área de busca adaptada para resolver o JSSP. Ye e Yan (2010) apresentaram uma estrutura cromossômica de duas linhas para um novo algoritmo genético baseado no procedimento de trabalho e distribuição da máquina.

Gonçalves *et al.* (2005) apresentaram um algoritmo genético híbrido para o JSSP, no qual a programação é construída usando uma regra de prioridade e as prioridades foram definidas pelo algoritmo genético, e então uma heurística de busca local foi aplicada para melhorar a solução. Zhang *et al.* (2005) propôs um algoritmo genético simulado para resolver o JSSP, nesse algoritmo, os autores combinam o GA e SA. Liu *et al.* (2006) combinaram o algoritmo genético tradicional com o método de Taguchi. Já Xu e Li (2007) propõe um algoritmo genético imune combinando a teoria imune e o algoritmo genético. Zhang e Wu (2010) combinaram o algoritmo TS e o algoritmo genético para desenvolver um algoritmo híbrido para o JSSP e também um algoritmo SA híbrido baseado em um novo mecanismo imunológico para o JSSP.

Apesar dos bons resultados obtidos nos artigos supracitados, observa-se que é salutar a utilização de algum método de refinamento de soluções em conjunto com o AG. Em geral, são aplicados métodos de busca local para refinar as soluções obtidas pelo algoritmo genético a cada geração.

Foi observado que a aplicação de técnica de busca local com uso do AG, otimiza soluções, de acordo com os resultados alcançados por Gonçalves *et al.* (2005), os testes aplicados atingiram o ótimo ou quase ótimo para todas as instâncias testadas.

Diversos outros autores desenvolveram métodos de busca local para otimizar os resultados do JSSP como nos trabalhos de Lourenço (1995), Vaessens *et al.* (1996), Lourenço e Zwijnenburg (1996) e Nowicki e Smutnicki (1996).

De fato, os algoritmos genéticos existentes para o JSSP apresentam geralmente uma velocidade de convergência lenta e fácil de capturar em soluções de

ótimo local. Para otimizar soluções, muitos pesquisadores concentraram sua atenção em combinar o algoritmo genético com esquemas de busca local para desenvolver algumas estratégias de otimização híbridas para o JSSP.

No trabalho de Gonçalves *et al.* (2005), os autores criam um algoritmo genético híbrido para tratar o JSSP e a representação do cromossomo é baseada em *Random Keys*. A abordagem deles foi combinar o algoritmo genético, um procedimento que gera o agendamento e um procedimento de busca local baseado nos modelos de Roy e Sussmann (1964) e Nowicki e Smutnicki (1996). Para os testes, utilizam 43 instancias de duas classes de problemas Fisher e Thompson (1963) FT06, FT10, FT20 e Lawrence (1984) LA01 a LA40 e compararam com outras metaheurísticas. O método proposto obteve melhor resultado em 31 instâncias mostrando melhor resultado na maioria das instâncias comparadas a outras metaheurísticas com exceção de Busca Tabu de Nowicki e Smutnicki (1996).

Wang (2012) propõe também um algoritmo híbrido para o JSSP, em que a estratégia é desenvolver um novo operador para o cruzamento (*crossover*) e para a mutação (*mutation*) para que haja uma diversidade populacional maior dos indivíduos evitando assim permanecer no ótimo local. Para os testes, o autor utilizou as mesmas instâncias de Gonçalves *et al.* (2005). O resultado do método adotado mostrou que houve melhor resultado em 33 instâncias das 43 testadas e os testes compraram também que o uso da Busca Local melhorou o resultado quando comparado a não utilização da Busca Local, comprovando a importância da Busca Local.

2.6. BUSCA LOCAL

Métodos de busca local realizam buscas na vizinhança de uma solução encontrada com o objetivo de encontrar soluções melhores. Exemplos de métodos de busca local frequentemente usados são Busca Tabu, *Simulated Annealing*, *Iterated Local Search* e o método de subida/descida de encosta, este último talvez a mais simples e popular estratégia de busca (PIZA-DAVILA *et al.*, 2017). Na Busca Tabu é definida uma transição simples de uma solução para outra. Iniciando com uma solução factível, ou seja, aceitável do problema, o procedimento continua criando e avaliando

soluções adicionais a partir de uma sequência de vizinhos. Em cada iteração, uma solução com o melhor *makespan* entre os vizinhos é selecionada. Além disso, para evitar que os espaços já visitados sejam recorrentes, movimentos realizados recentemente são mantidos em uma lista tabu para um número específico de iterações, conhecido como regime de acesso a tabu (SHEN, 2014). Segundo Glover (1989; 1990) uma técnica de busca tabu usa o histórico para lembrar das principais características das soluções recentemente visitadas e cria zonas do espaço de busca que se tornam proibidas (ou tabu).

O *Simulated Annealing* é um método que visa simular o resfriamento de metais. Para a busca *Simulated Annealing* o procedimento de verificar os melhores indivíduos é realizado de forma lenta, no qual o processo de aceitação de mudança do estado atual para assumir um determinado indivíduo é devido sua melhora, sempre que há melhor indivíduo, a movimentação é realizada. Inicialmente um estado de piora é aceito, mas conforme o número de iterações aumenta, o critério de aceitação de estados piores é reduzido. Segundo Linden (2012), se a diminuição de energia for realizada de forma lenta, como ocorre no processo de resfriamento de metais, o valor encontrado pelo algoritmo pode ser muito próximo do ótimo.

O *Iterated Local Search* o qual foi denotado inicialmente como “pontapé” (*Kick*), e posteriormente foi chamado de *Iterated Local Search* (ILS), sendo uma metaheurística simples e eficiente (LOURENÇO *et al.*, 2003). O conceito do ILS é de realizar o processo de perturbação na solução encontrada, para escapar de mínimos locais, mantendo assim as boas propriedades desse mínimo local (TANG, 2013). O ILS explora uma sequência de soluções criadas como perturbações da melhor solução atual, cujo resultado é refinado usando uma heurística embutida.

Existem outros métodos de busca que são conhecidos na literatura, como a subida da encosta (*Hill climbing*), descida de encosta (*Downhill*), a busca em vizinhança variável (VNS, do inglês *Variable Neighborhood Search*) e GRASP (*Greedy Randomized Adaptative Search Procedure*), entre outros. Destaca-se neste trabalho para o método de descida de encosta, o qual é baseado na busca em profundidade e parte do princípio de que deve-se atingir a solução no menor número possível de passos (CEYLAN e CEYLAN, 2012).

2.6.1. Método de Subida/Descida de Encosta

O método de subida/descida de encosta é um dos mais antigos algoritmos de busca e otimização (WEISE, 2009). O método realiza um loop no qual a melhor solução corrente conhecida p^* é usada como uma solução pai para produzir uma nova solução p_{new} . Se a nova solução é melhor que a solução pai, esta é substituída. O algoritmo de subida/descida de encosta é muito semelhante ao algoritmo genético. Embora muitas vezes o espaço de busca G e o espaço problemático sejam os mesmos dentre eles, o que diferencia o algoritmo de subida/descida de encosta é a generalidade. A primeira solução que é gerada através do método de subida/descida de encosta geralmente não usa parâmetros na operação de pesquisa e, a partir de então, operações unárias são geradas para a solução.

Sem perda de generalidade são usadas as operações de reprodução a partir do AG que escolhem os melhores indivíduos.

Pode-se ter problemas com o método de subida/descida de encosta, pois fica facilmente preso em um ótimo local. Sempre é usado a solução candidata mais conhecida, para encontrar novos pontos no espaço do problema. Subida/Descida de encosta utiliza uma operação de reprodução unária similar a mutação no AG. É observado que subida/descida de encosta pode ser implementado de forma determinística para o vizinho encontrado no espaço de busca G , que aqui é, na maioria das vezes, igual ao problema de espaço X , são sempre finitos e pode repetido. A Figura 17 apresenta o algoritmo do método de subida de encosta segundo Weise (2009).

Figura 17 - Método de subida de encosta

Algoritmo : $x^* \leftarrow \text{SubidaEncosta}(f)$	
<hr/>	
Ent.	: f : Função objetivo para minimização
Dado	: p_{new} : Nova solução criada
Dado	: p^* : Melhor indivíduo atual
Saída	: x^* : Melhor solução encontrada
<hr/>	
1	Início
2	$p^*.g \leftarrow \text{criado}()$
3	enquanto $\text{fimCritério}()$ faça
4	$p_{new}.g \leftarrow \text{mutação}(p^*.g)$
5	se $f(p_{new}.x) < f(p^*.x)$ faça $p^* \leftarrow p_{new}$
6	$p^*.x$
7	Fim

Fonte: Weise (2009)

O método de subida de encosta é empregado para maximização de um ponto no espaço de busca, portanto neste trabalho é utilizado o método de descida de encosta (downhill), no qual tem como objetivo a minimização de um ponto no espaço de busca Weise (2009).

2.6.2. Definição do conceito de vizinhança de uma solução

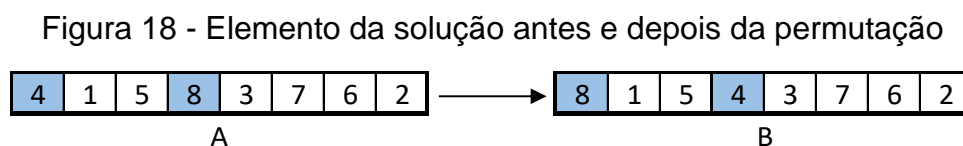
Em todos os métodos de busca local definidos na seção anterior deve ser definida, além da estratégia de busca, um conceito de vizinhança de uma solução. É importante destacar que definição da vizinhança exerce um significativo impacto nos métodos de busca local. Se o número de vizinhos de uma solução for muito grande o tempo computacional também tende a ser alto, porém aumenta-se as chances de encontrar uma boa solução. Por outro lado, se a vizinhança for muito pequena, pode-se não encontrar soluções boas. Portanto, uma correta definição da vizinhança pode facilitar a busca pela solução ótima.

Destaca-se a seguir alguns conceitos de vizinhança usados neste trabalho, os quais são classificados em vizinhanças diretas e indiretas. Nas vizinhanças diretas, o

conceito de vizinho é definido a partir de pequenas perturbações diretamente nas soluções do problema. Por outro lado, as vizinhanças diretas são obtidas perturbando uma representação indireta da solução.

2.6.3. Abordagem direta por permutação

Na busca local em vizinhança por permutação, a vizinhança é definida pela mudança de cada par de elementos da solução. São escolhidos os elementos da solução em pares aleatoriamente, portanto é realizada a troca de posição entre eles, gerando assim uma nova solução que é avaliada se é melhor do que a solução atual. Essa permutação ocorre com todos os elementos da solução. No exemplo apresentado da figura 18 é observado que os pares destacados sofrem permutação, os elementos de posição 1 e 4. O elemento 4 da posição 1 da solução A, assume a posição 4 na solução B e o elemento 8 da posição 4 da solução A, assume a posição 1 na solução B. Então é gerada uma solução B.



Fonte: o Autor

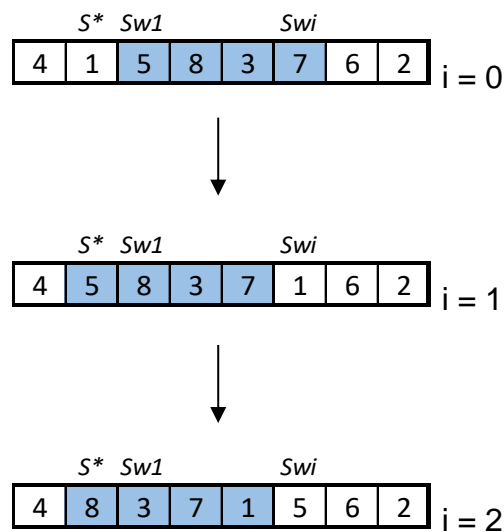
No caso de uma solução para um JSSP com j tarefas por m máquinas, por exemplo, a vizinhança é formada por $\binom{m \times j}{2}$ vizinhos, que significa a combinação dos elementos da solução 2 a 2.

2.6.4. Abordagem direta por inserção

No método de busca com abordagem direta por inserção uma janela na solução é definida aleatoriamente $W = (S_{ws}; S_{wf})$. Para cada elemento S_{wi} na janela, uma posição aleatória é escolhida antes ou depois da janela, chamada S^* . Caso o S^* seja menor do que o elemento S_{ws} , os elementos do intervalo (S^*, S_{ws}) são deslocados uma posição para a esquerda e S^* assume a posição de S_{wi} .

A Figura 19 faz uma alusão da mutação realizada nos elementos da solução quando o S^* é menor do que o elemento S_{ws} .

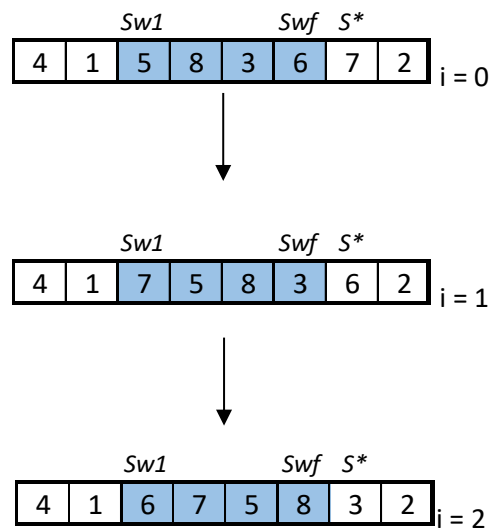
Figura 19 - Elementos da solução de abordagem direta por inserção $S^* < S_{ws}$



Fonte: o Autor

Se o elemento S^* for maior que o elemento S_{wf} , todos os genes do intervalo são deslocados para uma posição para direita, mas o elemento S^* vai para a posição S_{wi} . Ainda usando a figura 19 é apresentada a abordagem direta por inserção quando S^* é maior que o elemento S_{wf} .

Figura 20 - Elementos da solução de abordagem direta por de inserção $S^* > Swf$



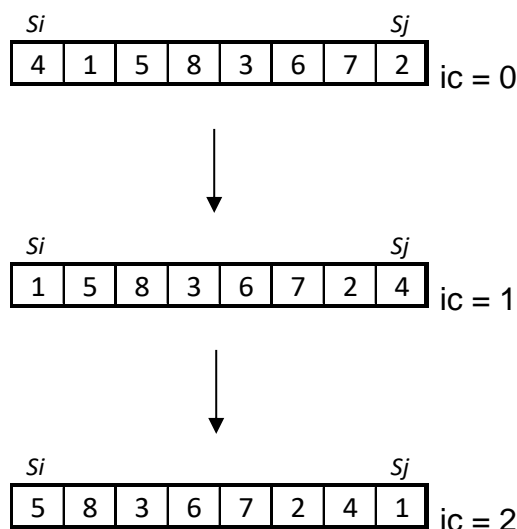
Fonte: o Autor

O número de vizinhos é igual ao tamanho da janela escolhida, então é escolhido o melhor vizinho caso exista, então o processo é repetido para o vizinho escolhido. Caso não encontre melhor vizinho o processo é encerrado (ARAÚJO *et al.*, 2009).

2.6.5. Abordagem direta por inserção completa

O conceito de vizinhança por inserção completa, segundo Araujo *et al.* (2009), foi derivado do método de busca por inserção, no qual cada um dos genes do intervalo (S_i, S_j) é movido uma posição para esquerda, e o gene S_j tem sua posição assumida pelo gene S_j como apresentado na figura 21.

Figura 21 - Elementos da solução de abordagem direta por inserção completa

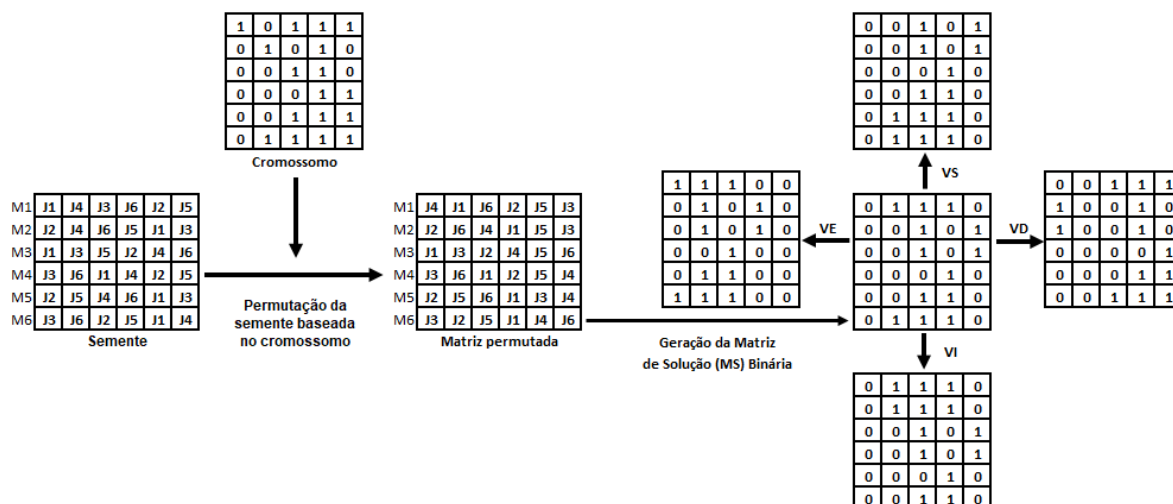


Fonte: o autor

2.6.6. Abordagem direta

A abordagem direta consiste em definir o vizinho de busca realizando pequenos distúrbios locais em uma representação direta da solução. No caso de um cromossomo binário bidimensional, no contexto do algoritmo genético, a perturbação é aplicada diretamente na solução do problema deslocando os valores dos genes horizontalmente e verticalmente. Na figura 22 é apresentada a abordagem direta com deslocamento vertical superior, no qual é gerado o Vizinho Superior (VS) a partir da Matriz de Solução (MS), é deslocada todas as linhas uma unidade para cima. No deslocamento vertical inferior, no qual é gerado o Vizinho Inferior (VI) a partir da MS, é deslocada todas as linhas uma unidade para baixo. No deslocamento horizontal para a direita, no qual é gerado o Vizinho Direita (VD) a partir da MS, é deslocada todas as linhas uma unidade para a direita. No deslocamento horizontal para a esquerda, no qual é gerado o Vizinho Esquerda (VE) a partir da MS, é deslocada todas as linhas uma unidade para a esquerda. Assim, cada solução MS tem apenas quatro vizinhos (LIMA, 2015), conforme exemplo da Figura 22. Esse conceito de vizinhança foi baseado no trabalho de Lima (2015).

Figura 22 - Conceito de abordagem direta



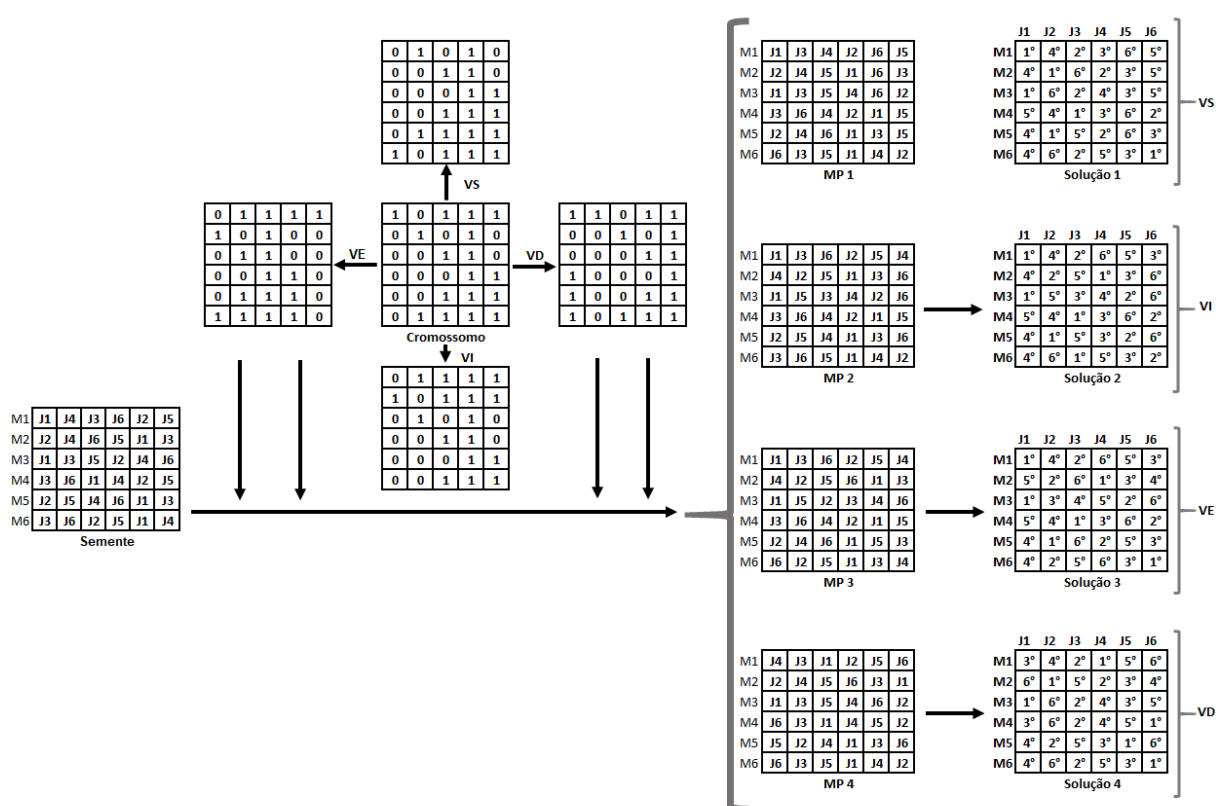
Fonte: o autor

2.6.7. Abordagem indireta

A abordagem indireta consiste em definir o vizinho de busca realizando pequenos distúrbios locais em uma representação indireta da solução. No caso de um cromossomo binário bidimensional, no contexto do algoritmo genético, a perturbação é aplicada ao cromossomo deslocando os valores dos genes horizontalmente e verticalmente. Na figura 23 é apresentada a abordagem indireta com deslocamento vertical superior, no qual é gerado o Vizinho Superior (VS) a partir do cromossomo, é deslocada todas as linhas uma unidade para cima, é gerada a matriz permutada 1 (MP1) a partir da matriz binária VS, portanto é gerada a solução 1 a partir da MP1. No deslocamento vertical inferior, no qual é gerado o Vizinho Inferior (VI) a partir do cromossomo, é deslocada todas as linhas uma unidade para baixo, é gerada a matriz permutada 2 (MP2) a partir da matriz binária VI, portanto é gerada a solução 2 a partir da MP2. No deslocamento horizontal para a esquerda, no qual é gerado o Vizinho Esquerda (VE) a partir do cromossomo, é deslocada todas as linhas uma unidade para a esquerda, é gerada

a matriz permutada 3 (MP3) a partir da matriz binária VE, portanto é gerada a solução 3 a partir da MP3. No deslocamento horizontal para a direita, no qual é gerado o Vizinho Direita (VD) a partir do cromossomo, é deslocada todas as linhas uma unidade para a direita, é gerada a matriz permutada 4 (MP4) a partir da matriz binária VD, portanto é gerada a solução 4 a partir da MP4.

Figura 23 - Conceito de abordagem indireta



Fonte: o autor

3. MÉTODOS E MATERIAIS

Neste capítulo são descritos os procedimentos metodológicos adotados na pesquisa, bem como os materiais utilizados.

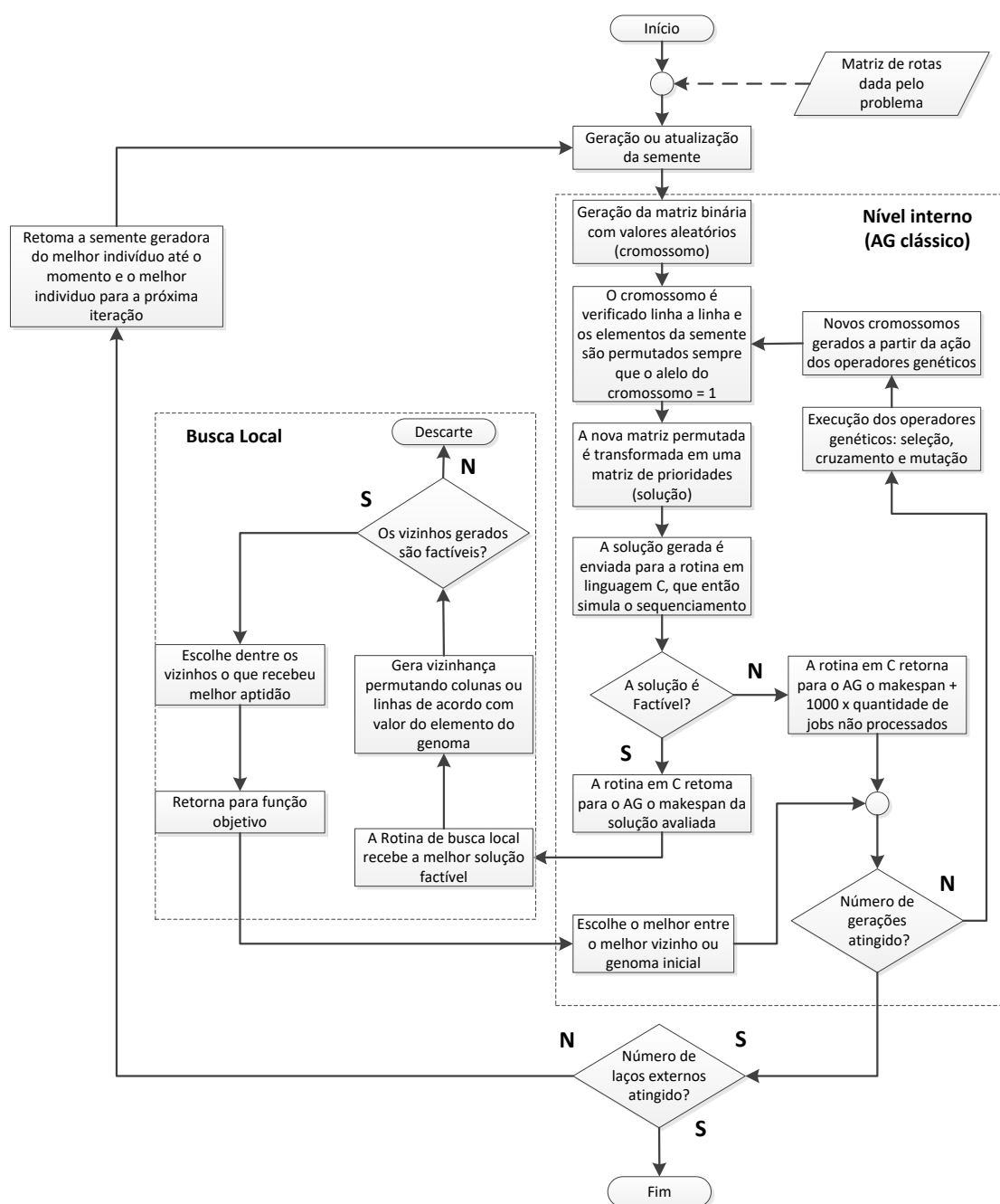
Uma das primeiras etapas do trabalho foi o estudo e a descrição de métodos de busca local comumente utilizados em conjunto com o Algoritmo Genético, para o problema de *Job Shop Scheduling*. Foi dada a prioridade para abordagens simples e relativamente rápidas, já que o custo computacional é um dos problemas desses métodos de busca local. Para tal, foi realizada uma revisão da literatura de forma exploratória, para um melhor entendimento do problema de pesquisa (GIL, 2008). Nessa etapa o método de descida de encosta foi escolhido como a estratégia de busca a ser utilizada.

Quanto aos critérios de vizinhança, quatro diferentes abordagens foram utilizadas: abordagem indireta e abordagem direta por permutação, por inserção e por inserção completa, todas definidas na seção 2.6. Importante destacar que os termos ‘direta’ e ‘indireta’ são usados aqui para definir o momento de aplicação da busca local, em referência à representação utilizada pelo Algoritmo Genético.

Este trabalho adota a vizinhança baseada em representação cromossômica indireta. Assim, a busca local pode ser aplicada ao genoma binário e, portanto, de forma indireta; ou ser aplicada após a decodificação do genoma diretamente na solução. Essas formas de aplicação da estratégia de busca estão ilustradas nas Figuras 22 e 23. Em resumo, tem-se a seguinte ordem de operações na busca local:

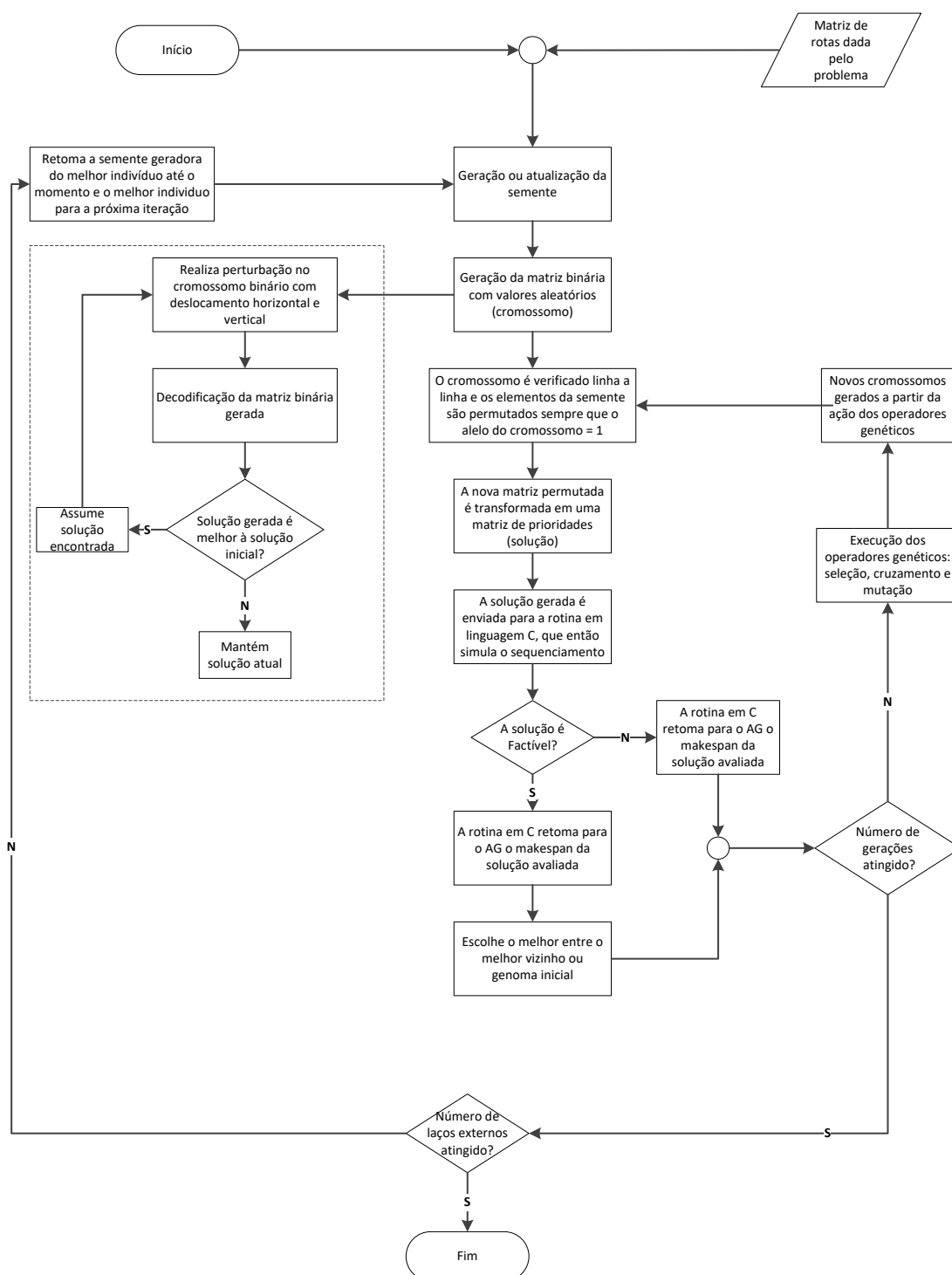
- 1) Abordagem direta - decodifica-se o cromossomo binário e define a vizinhança a partir da solução decodificada;
- 2) Abordagem indireta – define-se a vizinhança a partir do cromossomo binário e decodifica-se cada um dos indivíduos vizinhos.

Figura 24 - Fluxograma do AG com abordagem direta



Fonte: O autor (adaptado de Grassi, 2014)

Figura 25 - Fluxograma do AG com abordagem indireta



Fonte: o autor

Seguindo uma metodologia de teste, os critérios de vizinhança foram testados para avaliar sua influência no desempenho do AG com representação direta. O desempenho foi avaliado em relação à minimização do *makespan* e ao tempo computacional.

Em relação ao algoritmo genético, foram adotados os parâmetros definidos na Tabela 2.

Tabela 2 - Parâmetros adotados no algoritmo genético

Parâmetro	Descrição	Valor	Referência
População	Número de cromossomos binários na população	200	Experimental
Gerações	Número de gerações no AG	50	Experimental
Taxa de mutação	Probabilidade da mutação ser aplicada aos indivíduos	5%	Experimental
Taxa de cruzamento	Probabilidade de aplicação do operador de cruzamento	90%	Experimental
<i>IterOut</i>	Número de iterações no nível externo	200	Experimental

Fonte: o Autor

Os resultados obtidos com as estratégias de busca local são comparados com as abordagens apresentadas por Abdelmaguid (2010) e com o trabalho original de Grassi (2014) no qual a representação por semente dinâmica foi proposta e utilizada sem qualquer estratégia de busca local. Esses trabalhos foram escolhidos por apresentarem apenas abordagens baseadas puramente no AG sem considerar, portanto, abordagens híbridas baseadas no AG.

3.1. MATERIAIS

Neste trabalho foi utilizada a linguagem de programação C++, a biblioteca GAlib Wall (1996). A GAlib fornece um conjunto de objetos para programação do AG com uma variedade de representações e operadores genéticos. Essa biblioteca foi escolhida pela confiabilidade, em função do grande número de usuários, e pela vasta documentação.

Foi utilizado um computador com as seguintes configurações: Intel Core I5-M450 2.4Ghz, 8 GB de memória RAM, e sistema operacional Microsoft Windows 10, ambiente de desenvolvimento com a IDE *Code Blocks* 16.0.1, compilador GNU GCC *Compiler*.

Para obter os resultados, utilizou-se instâncias clássicas da literatura que foram definidas por Lawrence (1984). Dessa classe de instâncias foram utilizadas apenas as instâncias LA01 a LA20. Após essa etapa, os resultados são comparados com a literatura para verificar a eficiência do método proposto.

4. RESULTADOS E DISCUSSÕES

A seguir neste capítulo são apresentados os resultados alcançados nos experimentos, considerando o *makespan* encontrado para cada instância, considerando os problemas do LA01 ao LA20.

Este capítulo está dividido em duas partes: primeiramente apresenta-se os resultados obtidos com as diferentes estratégias de busca local. Na sequência, é apresentada uma comparação dos melhores resultados obtidos com abordagens da literatura.

4.1. CONFIGURAÇÃO DOS PARÂMETROS

Neste item são descritas as configurações dos parâmetros utilizados na implementação.

A composição da população inicial no AG é igual a $P = 200$ cromossomos e que passa pelo número de iterações $\text{Outlier} = 200$.

Foi utilizado $P_z = 90\%$ para o operador de crossover e para o operador de mutação foi utilizado $P_{\text{mut}} = 5\%$, sendo possível ocorrer mutação.

4.2. COMPARAÇÃO ENTRE AS ESTRATÉGIAS DE BUSCA LOCAL

Tabela 3 - Comparação de makespan para as diferentes estratégias de busca

Problema	Instância (Job X Máquina)	Número de Operações	Makespan Conhecido	AG + Busca Local	Makespan Encontrado
LA01	10 X 5	50	666	Indireta	666
				Permutação	710
				Inserção	715
				Inserção Completa	756
LA02	10 X 5	50	655	Indireta	669
				Permutação	710
				Inserção	734
				Inserção Completa	736

Tabela 3 - Comparação de makespan para as diferentes estratégias de busca
(Cont.)

Problema	Instância (Job X Máquina)	Número de Operações	Makespan Conhecido	AG + Busca Local	Makespan Encontrado
LA03	10 X 5	50	597	Indireta	617
				Permutação	653
				Inserção	654
				Inserção Completa	665
LA04	10 X 5	50	590	Indireta	607
				Permutação	692
				Inserção	699
				Inserção Completa	699
LA05	10 X 5	50	593	Indireta	593
				Permutação	593
				Inserção	593
				Inserção Completa	593
LA06	15 X 5	75	926	Indireta	926
				Permutação	926
				Inserção	926
				Inserção Completa	926
LA07	15 X 5	75	890	Indireta	890
				Permutação	953
				Inserção	979
				Inserção Completa	1012
LA08	15 X 5	75	863	Indireta	863
				Permutação	873
				Inserção	873
				Inserção Completa	899
LA09	15 X 5	75	951	Indireta	951
				Permutação	951
				Inserção	951
				Inserção Completa	951
LA10	15 X 5	75	958	Indireta	958
				Permutação	958
				Inserção	958
				Inserção Completa	958
LA11	20 X 5	100	1222	Indireta	1292
				Permutação	1292
				Inserção	1292
				Inserção Completa	1292

Tabela 3 - Comparação de makespan para as diferentes estratégias de busca
(Cont.)

Problema	Instância (Job X Máquina)	Número de Operações	Makespan Conhecido	AG + Busca Local	Makespan Encontrado
LA12	20 X 5	100	1039	Indireta	1039
				Permutação	1039
				Inserção	1039
				Inserção Completa	1039
LA13	20 X 5	100	1150	Indireta	1150
				Permutação	1150
				Inserção	1150
				Inserção Completa	1150
LA14	20 X 5	100	1292	Indireta	1292
				Permutação	1292
				Inserção	1292
				Inserção Completa	1292
LA15	20 X 5	100	1207	Indireta	1207
				Permutação	1249
				Inserção	1438
				Inserção Completa	1438
LA16	10 X 10	100	945	Indireta	998
				Permutação	1026
				Inserção	1028
				Inserção Completa	1028
LA17	10 X 10	100	784	Indireta	1004
				Permutação	1004
				Inserção	1028
				Inserção Completa	1022
LA18	10 X 10	100	848	Indireta	1016
				Permutação	1016
				Inserção	1017
				Inserção Completa	1040
LA19	10 X 10	100	842	Indireta	1015
				Permutação	995
				Inserção	1022
				Inserção Completa	1040
LA20	10 X 10	100	902	Indireta	986
				Permutação	1010
				Inserção	1023
				Inserção Completa	1015

Fonte: o autor

Os resultados da aplicação das estratégias de busca local aos problemas LA01 a LA20 são apresentados na Tabela 3. É possível observar que para diversas instâncias o *makespan* ótimo foi alcançado, como é o caso do LA01, LA05, LA06, LA07, LA08, LA09, LA10, LA12, LA13, LA14 e LA15, ou seja, 11 de 20 instâncias do problema testado neste trabalho atingiram o *makespan* conhecido na literatura. Vale destacar que o tempo computacional para a abordagem indireta foi muito menor comparado à busca por permutação, inserção e inserção completa. Portanto, mesmo as buscas que não alcançaram o melhor *makespan* chegaram próximo do ótimo. A abordagem indireta que tem a mutação realizada no cromossomo binário e que gera apenas 4 soluções vizinhas, apresentam menor tempo computacional em comparação as demais representações em que é gerada um número maior de vizinhança, e que portanto o tempo computacional aumenta consideravelmente.

Por outro lado, é importante destacar que para a abordagem indireta, cada vizinho deve ser decodificado para então avaliar a solução.

COMPARAÇÃO COM RESULTADOS DA LITERATURA

Nesta seção é apresentada uma comparação dos resultados de *makespan* obtidos com os trabalhos de Abdelmaguid (2010) e Grassi (2014). No trabalho de Abdelmaguid (2010) são apresentados resultados de *makespan* para diferentes representações do AG, a saber: representação baseada em ordem (OB), chaves aleatórias (RK), lista de preferência (PL), regras de prioridade (PR), baseada em máquina (MB), baseada em *job* (JB). No trabalho de Grassi (2014) foi proposta e testada a representação por semente dinâmica (SD). Os resultados são apresentados na Tabela 4, na qual destacam-se o número de *jobs* (*j*), máquinas (*m*) e de operações em cada instância do problema (*#*), e o melhor valor de *makespan* conhecido na literatura (BKM).

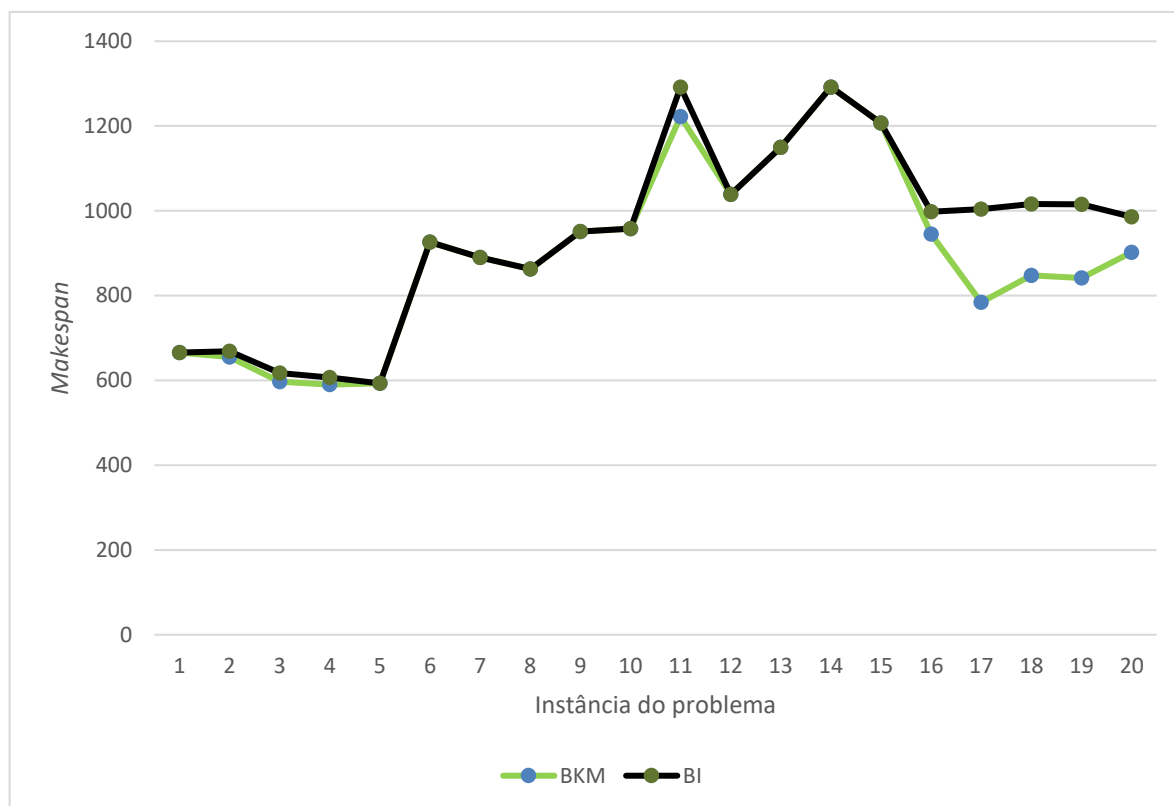
Tabela 4 - Comparação das melhores soluções obtidas neste trabalho com a literatura

Problemas Testados			Resultados								
Problema	$j \times m$	#	BKM	OB	RK	PL	PR	MB	JB	SD	BI
LA01	10 X 5	50	666	666	666	675	671	666	700	666	666
LA02	10 X 5	50	655	676	686	715	675	684	718	655	669
LA03	10 X 5	50	597	631	637	669	650	625	645	617	617
LA04	10 X 5	50	590	607	614	633	629	590	675	607	607
LA05	10 X 5	50	593	593	593	593	593	593	605	593	593
LA06	15 X 5	75	926	926	926	926	926	926	941	926	926
LA07	15 X 5	75	890	947	910	931	894	890	903	890	890
LA08	15 X 5	75	863	863	863	895	866	863	905	863	863
LA09	15 X 5	75	951	951	951	951	951	951	1009	951	951
LA10	15 X 5	75	958	958	958	958	958	958	987	958	958
LA11	20 X 5	100	1222	1222	1222	1242	1222	1222	1264	1222	1292
LA12	20 X 5	100	1039	1039	1039	1088	1039	1039	1069	1039	1039
LA13	20 X 5	100	1150	1150	1150	1189	1150	1150	1213	1150	1150
LA14	20 X 5	100	1292	1292	1292	1292	1292	1292	1300	1292	1292
LA15	20 X 5	100	1207	1274	1303	1390	1274	1207	1294	1212	1207
LA16	10 X 10	100	945	1014	1021	1078	1003	994	1080	1016	998
LA17	10 X 10	100	784	820	816	906	822	792	868	807	1004
LA18	10 X 10	100	848	933	928	977	901	857	986	894	1016
LA19	10 X 10	100	842	937	910	956	892	869	980	908	1015
LA20	10 X 10	100	902	989	1035	958	944	941	980	980	986
Média			896	924,4	926	951,1	917,6	905,45	956,1	912,3	936,95

Fonte: O autor

Alternativamente, os resultados de *makespan* são avaliados graficamente na Figura 26, considerando o *gap* obtido por cada método. O *gap* é definido aqui como a diferença entre a melhor solução obtida em cada caso e o melhor valor conhecido na literatura para cada instância do problema.

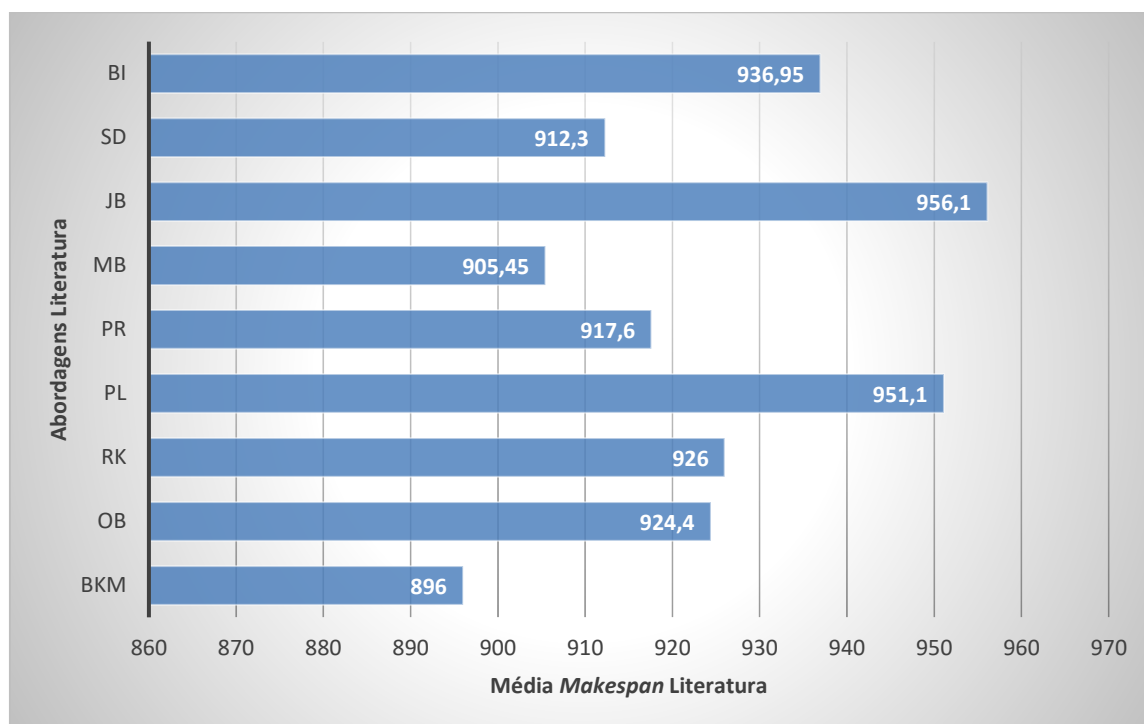
Figura 26 - Comparação dos melhores makespan obtidos com os resultados alcançados por abordagem indireta.



Fonte: o autor

Na figura 27 pode-se observar as médias de *makespan* alcançados pelas diferentes abordagens, com vistas a fornecer uma visão global dos métodos. Os valores nas linhas horizontais representam a média por representação e que tem como base a tabela 3. Observa-se que a melhor solução obtida neste trabalho apresentou um resultado pela média, superior à representação baseada em lista (*PL*) e baseado em *job* (*JB*). Já para outras representações, como *Randon Key* (*RK*) e *Operation-Based* (*OB*) ficou próximo da média.

Figura 27 - Média de makespan alcançado comparado à outras representações



Fonte: O autor.

Na tabela 5 são apresentados os valores mínimos, médias e máximo de *makespan* obtido para cada instância do problema, considerando do LA01 ao LA20. Os valores destacados em negrito representam o mínimo alcançado após a execução de 200 iterações para cada instância. É observado que para algumas instâncias os valores máximos, médios e mínimos são iguais, o que demonstra a robustez do método aplicado.

Tabela 5 - Valores mínimos, médias e máximos de makespan obtido pela abordagem indireta

	LA01	LA02	LA03	LA04	LA05	LA06	LA07	LA08	LA09	LA10
Máximo	858	904	775	854	629	1015	1096	1102	1024	958
Média	671	678	622	629	593	927	901	867	952	958
Mínimo	666	669	617	607	593	926	890	863	951	958

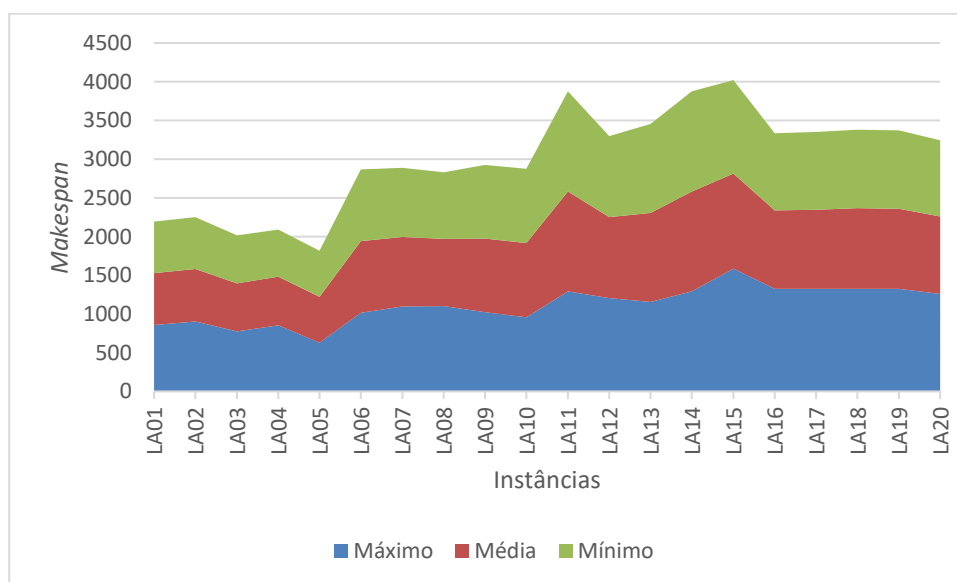
Tabela 5 – Valores mínimos, médios e máximos de *makespan* obtidos pela abordagem indireta (Cont.)

	LA11	LA12	LA13	LA14	LA15	LA16	LA17	LA18	LA19	LA20
Máximo	1292	1205	1154	1292	1586	1327	1327	1327	1327	1259
Média	1292	1048	1150	1292	1228	1010	1020	1039	1031	1000
Mínimo	1292	1044	1150	1292	1207	998	1004	1016	1015	986

Fonte: O autor

Na figura 28 também são apresentados os valores mínimos, médios e máximos alcançados, porém em forma de gráfico para facilitar a visualização.

Figura 28 - Gráfico dos valores mínimos, médios e máximos de makespan obtidos



Fonte: O autor.

5. CONCLUSÃO

A dissertação apresentada buscou atender o objetivo proposto inicialmente, que foi de aplicar métodos de busca local para o problema JSSP, em conjunto com o AG, com vistas a avaliar a influência de conceitos de vizinhanças definidos a partir de diferentes representações da solução, sendo indireta e direta no AG no contexto do JSSP.

Os resultados obtidos, que tem por base o método de descida de encosta, mostram que o método de busca local com abordagem direta encontra boas soluções, chegando ao ótimo em algumas instâncias, porém com um elevado tempo computacional, devido ao processo de permutação ou inserção na vizinhança para as soluções já existentes. Já os métodos com abordagem indireta, apresentam que o tempo computacional é consideravelmente menor comparado a abordagem direta e que também alcança o ótimo da literatura para várias instâncias dos problemas, portanto é observado que a correta definição da representação utilizada pode determinar as soluções encontradas em tempos computacionais aceitáveis.

Como sugestões futuras, pode-se aplicar os conceitos utilizados neste trabalho em um único método, como por exemplo uma busca em vizinhança variável. Ainda como sugestão de trabalho futuro, pode-se testar outros métodos de busca local, diferente do método utilizado de descida de encosta.

REFERÊNCIAS

- ADAMS, J.; BALAS, E.; ZAWAK, D. The shifting bottleneck procedure for job shop scheduling. **Management science**, v. 34, n. 3, p. 391-401, 1988.
- APPLEGATE, D.; COOK, W. A computational study of the job-shop scheduling problem. **ORSA Journal on computing**, v. 3, n. 2, p. 149-156, 1991.
- ARAUJO, R.; SANTOS, A.; ARROYO, J. Genetic Algorithm and Local Search for. **IEEE Congress on Evolutionary Computation**, p. 955-961, 2009.
- ARAÚJO, Sidnei Alves de; LIBRANTZ, André Felipe Henriques; ALVES, Wonder Alexandre Luz. Técnicas metaheurísticas aplicadas na otimização de parâmetros em um modelo probabilístico de gestão de estoques. **INGEPRO-Inovação, Gestão e Produção**, v. 2, n. 10, p. 013-022, 2010.
- ASADZADEH, Leila. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, v. 85, p. 376-383, 2015.
- BAKER, K.; FRENCH, S. Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop, 1983.
- BAPTISTE, P.; FLAMINI, M.; SOURD, F. Lagrangian bounds for just-in-time job-shop. **Computers & Operations Research**, v. 35, p. 906-915., 2008.
- BEAN, J. C. Genetic Algorithms and Random Keys for Sequencing and Optimization *INFORMS Journal on Computing*.
- BEN-DAYA, M.; AL-FAWZAN, M. A tabu search approach for the flow shop scheduling problem. **European journal of operational research**, v. 109, n. 1, p. 88-95, 1998.
- BISKUP, D. Single-machine scheduling with learning considerations. **European Journal of Operational Research**, v. 115, n. 1, p. 173-178, 1999.
- BORGES, Joao Paulo Veloso et al. Planejamento e controle da produção de uma indústria de cataventos apoiado pelo gráfico de gantt: um estudo de caso. XXXIII Encontro Nacional de Engenharia de Produção. Anais... Salvador: ABEPRO, 2013.

BRUCKER, P.; JURISCH, B.; SIEVERS, B. . A branch and bound algorithm for the job-shop scheduling problem. **Discrete applied mathematics**, v. 49, n. 1-3, p. 107-127, 1994.

CARLIER, J.; PINSON, É. An algorithm for solving the job-shop problem. **Management science**, v. 35, n. 2, p. 164-176, 1989.

CEYLAN, Huseyin; CEYLAN, Halim. A hybrid harmony search and TRANSYT hill climbing algorithm for signalized stochastic equilibrium transportation networks. **Transportation Research Part C: Emerging Technologies**, v. 25, p. 152-167, 2012.

CORDENONSI, Andre Zanki. Ambientes, objetos e dialogicidade: uma estratégia de ensino superior em heurísticas e metaheurísticas. 2008.

DAI, M.; TANG, D.; GIRET, A.; SALIDO, M. A.; LI, W. D. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. **Robotics and Computer-Integrated Manufacturing**, v. 29, n. 5, p. 418-429, 2013.

DAVIS, L. Job shop scheduling with genetic algorithms. In: **Proceedings of an international conference on genetic algorithms and their applications**. [S.l.]: [s.n.], v. 140, 1985.

DONG, Fangpeng; AKL, Selim G. **Scheduling algorithms for grid computing: State of the art and open problems**. Technical report, 2006.

FISHER, H.; THOMPSON, G. L. Probabilistic learning combinations of local job-shop scheduling rules. **Industrial scheduling**, v. 3, n. 2, p. 225-251, 1963.

GAGNÉ, C.; PRICE, W. L.; GRAVEL, M. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. **Journal of the Operational Research Society**, v. 53, n. 8, p. 895-906, 2002.

GAO, J.; SUN, L.; GEN, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. **Computers & Operations Research**, v. 35, n. 9, p. 2892-2907, 2008.

GAO, K. Z.; SUGANTHAN, P. N.; PAN, Q. K.; CHUA, T. J.; CAI, T. X.; CHONG, C. S. Discrete harmony search algorithm for flexible job shop scheduling problem with

multiple objectives. **Journal of Intelligent Manufacturing**, v. 27, n. 2, p. 363-374, 2016.

GEN, M.; SYARIF, A. Hybrid genetic algorithm for multi-time period production/distribution planning. **Computers & Industrial Engineering**, n. 48, p. 799 - 809, 2005.

GENDREAU, M.; POTVIN, J.-Y. Handbook of Metaheuristics. [S.l.]: Springer New York Dordrecht Heidelberg London, v. 146, 2010. Cap. 12.

GIFFLER, B.; THOMPSON, G. L. Algorithms for solving production-scheduling problems. **Operations research**, v. 8, n. 4, p. 487-503, 1960.

GONÇALVES, J. F.; MENDES, J. M. A look-ahead dispatching rule for scheduling operations. **VII Latin-Ibero-American Conference on Operations Research and Systems Engineering**, Chile, 1994.

GONÇALVES, J. F.; MENDES, M. J. J.; RESENDE, M. G. A hybrid genetic algorithm for the job shop scheduling problem. **European journal of operational research**, v. 167, n. 1, p. 77-95, 2005.

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

GRASSI, F. SIMULAÇÃO E OTIMIZAÇÃO POR ALGORITMOS GENÉTICOS DO SEQUENCIAMENTO DE ORDENS DE PRODUÇÃO EM AMBIENTES JOB SHOP, São Paulo, 2014.

GRAY, C. F.; HOESADA, M. Matching heuristic scheduling rules for job shops to the business sales level. **Production and Inventory Management Journal**, v. 32, n. 2, p. 12, 1991.

HIRAISHI, K.; LEVNER, E.; VLACH, M. Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. **Computers & Operations Research**, v. 29, n. 7, p. 841-848, 2002.

HOLLAND, J. H.; HOLLAND, J. H. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. **Ann Arbor, MI: University of Michigan Press**, 1975.

JAIN, A. S.; MEERAN, S. **A state-of-the-art review of job-shop scheduling techniques**. Department of Applied Physics, Electronic and Mechanical Engineering University of Dundee. Dundee. 1998.

KIRKPATRICK, S. Optimization by Simulated Annealing.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, n. 220, p. 671-680, 1983.

KUO, W. H.; YANG, D. L. Minimizing the total completion time in a single-machine scheduling problem with a time-dependent learning effect. **European Journal of Operational Research**, v. 174, n. 2, p. 1184-1190, 2006.

LAGEWEG, B. J.; LENSTRA, J. K.; RINNOOY KAN, A. H. G. Job-shop scheduling by implicit enumeration. **Management Science**, v. 24, n. 4, p. 441-450, 1977.

LAGUNA, M.; BARNES, J. W.; GLOVER, F. W. Tabu search methods for a single machine scheduling problem. **Journal of Intelligent Manufacturing**, v. 2, n. 2, p. 63-73, 1991.

LAWRENCE, S. **Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques**. Carnegie Mellon University. Pittsburgh. 1984.

LEE, Y. H.; PINEDO, M. Scheduling jobs on parallel machines with sequence-dependent setup times. **European Journal of Operational Research**, v. 100, n. 3, p. 464-474, 1997.

LI, Xinyu; GAO, Liang. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. **International Journal of Production Economics**, v. 174, p. 93-110, 2016.

LIAW, C. F. A hybrid genetic algorithm for the open shop scheduling problem. **European Journal of Operational Research**, v. 124, n. 1, p. 28-42, 2000.

LIMA, S. J. A. OTIMIZAÇÃO DO PROBLEMA DE ROTEAMENTO DE VEÍCULOS CAPACITADO USANDO ALGORITMOS GENÉTICOS COM HEURÍSTICAS E REPRESENTAÇÕES CROMOSSÔMICAS ALTERNATIVAS, São Paulo, 2015.

LINDEN, Ricardo. **Algoritmos genéticos (2a edição)**. Brasport, 2008.

LINDEN, Ricardo. **Algoritmos genéticos (3a edição)**. Ciência Moderna, 2012.

LIU, T. K.; TSAI, J. T.; CHOU, J. H. Improved genetic algorithm for the job-shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, v. 27, n. 9-10, p. 1021-1029, 2006.

LOURENCO, H. R. Computational study of local search and large-step optimization methods. **European Journal of Operational Research**, v. 83, n. 2, p. 347-364, 1995.

LOURENÇO, H. R.; ZWIJNENBURG, M. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In: **Meta-Heuristics**. [S.l.]: Springer, 1996. p. 219--236.

LOURENÇO, H.; MARTIN, O.; STÜTZLE, J. Iterated local search. In: GLOVER, F.; KOCHENBERGER. Handbook of Metaheuristics. **New York: Springer**, n. 57, p. 320–353, 2003.

MATTFELD, D. C.; BIERWIRTH, C. An efficient genetic algorithm for job shop. **European journal of operational research**, v. 155, n. 3, p. 616-630, 2004.

MÓDOLO JUNIOR, Valdemar et al. Estudo comparativo de diferentes representações cromossômicas nos algoritmos genéticos em problemas de sequenciamento da produção em job shop. 2015.

NOWICKI, E.; SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. **Management science**, v. 42, n. 6, p. 797-813, 1996.

NOWICKI, E.; SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. **Management science**, v. 42, n. 6, p. 797-813, 1996.

OMBUKI, B. M.; VENTRESCA, M. Local search genetic algorithms for the job shop scheduling problem. **Applied Intelligence**, v. 21, n. 1, p. 99-109, 2004.

PARK, B. J.; CHOI, H. R.; KIM, H. S. A hybrid genetic algorithm for the job shop scheduling problems. **Computers & industrial engineering**, v. 45, n. 4, p. 597-613, 2003.

PEZZELLA, F.; MORGANTI, G.; CIASCETTI, G. A genetic algorithm for the flexible job-shop scheduling problem. **Computers & Operations Research**, v. 35, n. 10, p. 3202-3212, 2008.

PINEDO, M. **Scheduling, theory, algorithms, and systems**. 4. ed. New York: Englewood Cliffs, 2008.

PIZA-DAVILA, Ivan et al. A CUDA-based Hill-climbing Algorithm to Find Irreducible Testors from a Training Matrix. **Pattern Recognition Letters**, 2017.

PONGCHAIRERKS, P.; KACHITVICHYANUKUL, V. A two-level particle swarm optimisation algorithm for open-shop scheduling problem. **International Journal of Computing Science and Mathematics**, v. 7, n. 6, p. 575-585, 2016.

ROY, B.; SUSSMANN, B. Les problemes d'ordonnancement avec contraintes disjonctives. **Notes**, v. 9, 1964.

ROY, Provas Kumar. Teaching learning based optimization for short-term hydrothermal scheduling problem considering valve point effect and prohibited discharge constraint. **International Journal of Electrical Power & Energy Systems**, v. 53, p. 10-19, 2013.

RUIZ, R.; VÁZQUEZ-RODRÍGUEZ, J. A. The hybrid flow shop scheduling problem. **European Journal of Operational Research**, v. 205, n. 1, p. 1-18, 2010.

RUSSELL, Stuart; NORVIG, Peter; INTELLIGENCE, Artificial. A modern approach. **Artificial Intelligence. Prentice-Hall, Englewood Cliffs**, v. 25, p. 27, 1995.

SABUNCUOGLU, I.; BAYIZ, M. Job shop scheduling with beam search. **European Journal of Operational Research**, v. 118, n. 2, p. 390-412, 1999.

SANTOS, RENATO ALESSANDRO ROCHA. ANÁLISE DE REPRESENTAÇÕES CROMOSSÔMICAS PARA ALGORITMOS GENÉTICOS NA SOLUÇÃO DO PROBLEMA DE ROTEAMENTO DE VEÍCULOS.

SHA, D. Y.; HSU, C. Y. A new particle swarm optimization for the open shop scheduling problem. **Computers & Operations Research**, v. 35, n. 10, p. 3243-3261, 2008.

SHEN, L. A tabu search algorithm for the job shop problem with sequence. **Computers & Industrial Engineering**, Germany, Setembro 2014.

SILVEIRA, Sidnei Renato; BARONE, Dante Augusto Couto. Jogos Educativos computadorizados utilizando a abordagem de algoritmos genéticos. **Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciências da Computação**, 1998.

SOMEYA, H.; YAMAMURA, M. Genetic algorithm with search area adaptation for the function optimization and its experimental analysis. In: **Evolutionary Computation, 2001. Proceedings of the 2001 Congress on**. [S.l.]: IEEE, v. 2, 2001. p. 933--940.

VAESSENS, R. J. M.; AARTS, E. H.; LENSTRA, J. K. Job shop scheduling by local search. **INFORMS Journal on Computing**, v. 8, n. 3, p. 302-317, 1996.

VILCOT, G.; BILLAUT, J. C. A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. **European Journal of Operational Research**, v. 190, n. 2, p. 398-411, 2008.

WALL, Matthew. GAlib: A C++ library of genetic algorithm components. **Mechanical Engineering Department, Massachusetts Institute of Technology**, v. 87, p. 54, 1996.

WANG, H. Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. **Expert Systems**, v. 22, n. 2, p. 78-85, 2005.

WANG, L.; ZHENG, D. Z. An effective hybrid optimization strategy for job-shop scheduling problems. **Computers & Operations Research**, v. 28, n. 6, p. 585-596, 2001.

WANG, Y. A new hybrid genetic algorithm for job shop scheduling problem. **Computers & Operations Research**, v. 39, n. 10, p. 2291-2299, 2012.

WATANABE, M.; IDA, K.; GEN, M. A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. **Computers & Industrial Engineering**, v. 48, n. 4, p. 743-752, 2005.

WEISE, Thomas. Global optimization algorithms-theory and application. **Self-published**, v. 2, 2009.

WILLIAMSON, D. P.; HALL, L. A.; HOOGEVEEN, J. A.; HURKENS, C. A.; LENSTRA, J. K.; SEVAST'JANOV, S. V. E.; SHMOYS, D. B. Short shop schedules. **Operations Research**, v. 45, n. 2, p. 288-294, 1997.

XU, X. D.; LI, C. X. Research on immune genetic algorithm for solving the job-shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, v. 34, n. 7, p. 783-789, 2007.

YAGMAHAN, B.; YENISEY, M. M. Ant colony optimization for multi-objective flow shop scheduling problem. **Computers & Industrial Engineering**, v. 54, n. 3, p. 411-420, 2008.

YE, L.; YAN, C. H. E. N. A genetic algorithm for job-shop scheduling. **Journal of software**, v. 5, n. 3, p. 269-274, 2010.

ZHANG, C.; LI, P.; RAO, Y.; LI, S. A new hybrid GA/SA algorithm for the job shop. In: **European Conference on Evolutionary Computation in Combinatorial Optimization**. Berlin: Springer, 2005. p. 246-259.

ZHANG, G.; GAO, L.; SHI, Y. An effective genetic algorithm for the flexible job-shop scheduling problem. **Expert Systems with Applications**, v. 38, n. 4, p. 3563-3573, 2011.

ZHANG, R.; WU, C. A hybrid immune simulated annealing algorithm for the job shop scheduling problem. **Applied Soft Computing**, v. 10, n. 1, p. 79-89, 2010.

ZHOU, Jian-Ming. Computability vs. Nondeterministic and P vs. NP. arXiv preprint arXiv:1305.4029, 2013.

ZHOU, H.; FENG, Y.; HAN, L. The hybrid heuristic genetic algorithm for job shop scheduling. **Computers & Industrial Engineering**, v. 40, n. 3, p. 191-200, 2001.