UNIVERSIDADE NOVE DE JULHO – UNINOVE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

VALDEMAR MÓDOLO JUNIOR

ESTUDO COMPARATIVO DE DIFERENTES REPRESENTAÇÕES CROMOSSÔMICAS NOS ALGORITMOS GENÉTICOS EM PROBLEMAS DE SEQUENCIAMENTO DA PRODUÇÃO EM *JOB SHOP*

São Paulo
2015

VALDEMAR MÓDOLO JUNIOR

ESTUDO COMPARATIVO DE DIFERENTES REPRESENTAÇÕES CROMOSSÔMICAS NOS ALGORITMOS GENÉTICOS EM PROBLEMAS DE SEQUENCIAMENTO DA PRODUÇÃO EM *JOB SHOP*

Dissertação de mestrado apresentado ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Nove de Julho – UNINOVE, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Produção.

Prof. Fabio Henrique Pereira, Dr. – Orientador.

São Paulo

2015

Módolo Junior, Valdemar.
Estudo comparativo de diferentes representações cromossômicas nos algoritmos genéticos em problemas de sequenciamento da produção em job shop. /Valdemar Módolo Junior 2015.
135 f.
Dissertação (mestrado) – Universidade Nove de Julho - UNINOVE, São Paulo, 2015.
Orientador (a): Prof. Dr. Fabio Henrique Pereira.
1. Representação cromossômica. 2. Operadores genéticos. 3. Sequenciamento da produção. 4. Job shop. 5. Algoritmos genéticos.
I. Pereira, Fabio Henrique. II. Titulo

CDU 658.5

UNINOVE
Universidade Nove de Julho

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE DISSERTAÇÃO

DE

VALDEMAR MÓDOLO JUNIOR

Título da Dissertação: ESTUDO COMPARATIVO DE DIFERENTES REPRESENTAÇÕES CROMOSSÔMICAS NOS ALGORITMOS GENÉTICOS EM PROBLEMAS DE SEQUENCIAMENTO DA PRODUÇÃO EM *JOB SHOP*.

A COMISSÃO EXAMINADORA, COMPOSTA PELOS PROFESSORES ABAIXO, CONSIDERA O CANDIDATO VALDEMAR MÓDOLO JUNIOR Aprovado.

São Paulo, 10 de junho de 2015.

Presidente: PROF. DR. FABIO HENRIQUE PEREIRA

Membro: PROF. DR. THIAGO ANTONIO GRANDI DE TOLOSA

Membro: PROF. DR. RENATO JOSÉ SASSI

Dedico este trabalho a minha esposa e filhos por todo apoio, paciência e carinho que me deram, ajudando a vencer mais esta etapa da vida, este grande desafio e sonho pessoal, a esta família maravilhosa, muito obrigado pela motivação e inspiração.

AGRADECIMENTOS

Agradeço ao Pai Eterno e mantenedor de tudo, pela sua graça e sabedoria que concede aos homens.

Toda jornada começa com o primeiro passo, sendo assim na jornada da vida cada passo é único. Até o momento posso dizer que muitos passos foram dados e muitas pessoas compartilharam comigo nesta jornada.

Agradeço a minha querida esposa, GIANE, pela colaboração, compreensão e apoio, sabemos que sacrifícios são necessários, quando a recompensa é boa, e compartilhar jornada torna tudo mais fácil.

Agradeço aos filhos MIGUEL, DANIEL e GABRIEL, pelo incentivo e a jovialidade que só as crianças e juvenis possuem.

Agradeço ao professor Dr. Fabio Henrique Pereira, pela paciência dedicação e coleguismo, sem a sua imensa ajuda não teríamos chegado ao final. Também agradeço a todos os professores do programa de Mestrado em Engenharia de Produção da UNINOVE, sem dúvida nestes dois anos aprendi muito !!!

Agradeço aos meus pais, pois mesmo sem compreenderem plenamente o grau de dificuldades que enfrentamos para a conclusão do mestrado, os mesmos têm torcido e se preocupado por mim.

E também agradeço a Sabesp e meu departamento pelo apoio e compreensão.

“Conhecereis a verdade e a verdade vos libertará. ”

(Jesus Cristo)

Resumo

Dentre os métodos de otimização, o Algoritmo Genético (AG) vem produzindo bons resultados em problemas com ordem de complexidade elevada, como é o caso, por exemplo, do problema de sequenciamento da produção em ambiente job shop. Os problemas de sequenciamento da produção devem ser traduzidos para uma representação matemática, para que o AG possa atuar. Neste processo surgi uma problemática, a escolha entre as diferentes formas de se representar a solução visto que algumas representações apresentam limitações, como apresentar soluções não factíveis e/ou redundantes. Portanto o objetivo deste trabalho é realizar um estudo comparativo entre diferentes representações da solução no AG em problemas de sequenciamento da produção em ambientes job shop. Duas representações da solução foram analisadas, a baseada em listas de prioridades e a baseada em ordem de operações e comparada com uma representação binária, no contexto do conjunto de problemas de sequenciamento definidos por Lawrence (1984). Os resultados foram avaliados em função do tempo total de processamento (makespan), do custo computacional e da proporção de soluções factíveis geradas. Percebeu-se que, a representação da solução baseada em ordem de operações, a qual produziu 100% de soluções factíveis, foi a que mostrou os melhores resultados apesar de não apresentar convergência para a melhor solução conhecida em todos os problemas.

Palavras chave: Representação cromossômica, Operadores Genéticos, Sequenciamento da produção, *job shop*, Algoritmos Genéticos.

Abstract

Among the optimization methods, the Genetic Algorithm (GA) has been producing good results in problems with high order of complexity, such as, for example, the production scheduling problem in job shop environment. The production sequencing problems must be translated into a mathematical representation, so that the AG can act. In this process we came up a problematic, the choice between different ways to represent the solution as some representations have limitations, how to present not feasible and / or redundant solutions. Therefore the aim of this study is to conduct a comparative study between different representations of the solution in the AG in production sequencing problems in job shop environments. Two representations of the solution were analyzed, the priority lists based and based on order of operations and compared with a binary representation, in the context of sequencing problem set defined by Lawrence (1984). The results were evaluated according to the total processing time (makespan), the computational cost and the proportion of generated feasible solutions. It was noticed that the representation of the solution based on order of operations, which produced 100% of feasible solutions, was the one that showed the best results although no convergence to the best known solution to every problem.

Key words: chromosomal representation, genetic operators, production sequencing, job shop, genetic algorithms.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS E TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABC – *Artificial Bee Colony*

ACO – *Ant Colony Optimization*

AE – *Algoritmos Evolutivos*

AG – *Algoritmos Genéticos*

CX – *Cycle Crossover*

FIFO – *First in, First out*

GAlib – *Genetic Algorithms Library*

GRASP – *Greedy Randomized adaptive Search Procedure*

JB – *Job-based representation*

JSSP – *Job Shop Scheduling Problems*

LA – *instância de problemas job shop*

LOX – *Linear Order Crossover*

MB – *Machine-based representation*

OB – *Operation-based representation*

OX – *Order Crossover*

PSO – *Particle Swarm Optimisation*

PMX – *Partially Mapped Crossover*

PPX – *Precedence Preservative Crossover*

PR – *Priority rule-based representation*

PL – *Preference list-based representation*

RK – *Random key representation*

SA – *Simulated annealing*

SD – *Semente Dinâmica*

VNS – *Variable Neighborhood Search*

SUMÁRIO

1 INTRODUÇÃO

A acelerada evolução tecnológica em diversas áreas do conhecimento no último século e a grande expansão das fronteiras comerciais e industriais entre as nações impõem a unificação de métodos, processos e mudanças no estilo de vida, influenciando o consumo global e individual.

Neste cenário competitivo surge a necessidade de a indústria buscar métodos e estratégias que levem a produtos e serviços de melhor qualidade, combinados com o menor custo e minimizando tempo gasto na produção. A otimização dos processos de produção é essencial e tem sido muito focada em diversas áreas do conhecimento como, por exemplo, gestão empresarial, economia, logística, entre outros e, mais particularmente no estudo da engenharia de produção (KUNNATHUR et al., 2004; HEINONEN; PETTERSSON, 2007).

Todo o processo de otimização parte do conhecimento geral ao específico, visando melhorar a disponibilidade dos recursos existentes, muitas vezes, escassos. Nos últimos cinquenta anos os pesquisadores buscam novas abordagens, ou aperfeiçoar as existentes, relativas à otimização dos processos de produção em ambientes industriais.

Um dos grandes problemas da engenharia de produção é a otimização do sequenciamento ou escalonamento da produção. Otimização consiste em identificar a melhor sequência de atendimento na linha de produção, que minimize, entre outros critérios, o tempo total de produção, chamado de *makespan*, reduzindo ao máximo o tempo de ociosidade, bem como a melhor disposição das várias máquinas na produção.

O problema de sequenciamento da produção, em especial em ambientes de produção do tipo *job shop*, consiste num conjunto de tarefas (*job*) invariavelmente processadas em todas as máquinas disponíveis e segue uma rota pré-estabelecida que difere de um *job* para outro (FAN; ZHANG, 2010). Atualmente é um dos mais difíceis de resolver, devido a sua complexidade computacional, natureza combinatória e demais restrições inerentes a esse tipo de ambiente produtivo.

De uma forma geral, o grau de complexidade do problema de sequenciamento da produção tende a aumentar com o crescimento do número de máquinas envolvidas e operações efetuadas. Para problemas com elevada complexidade computacional, os algoritmos genéticos (AG), tem produzido bons resultados em relação a outros métodos de otimização. Define-se AG como algoritmo computacional baseado nos conceitos da teoria da evolução das espécies, que emita a troca das características de cada indivíduo na geração de uma nova prole, aplicando-se conceitos de cruzamento genético, mutação, etc.

Como todo problema de otimização, o sequenciamento da produção deve ser traduzido em uma representação matemática para que o AG possa atuar. Nesse processo, um dos pontos críticos é como representar o indivíduo ou cromossomo no AG, para o problema um cromossomo representa a sequência de uma solução específica. Sendo assim, a representação cromossômica no AG é a representação da solução do problema real. Na literatura encontram-se diferentes formas de representar uma solução do sequenciamento em ambientes *job shop*, (GEN et al. 1994; KUBOTA, 1995; FAN et al 1993; DAVIS, 1985; GRASSI, 2014; BEAN,1994; GONÇALVES; RESENDE; RUIZ et al., 2015).

Vale destacar que por representação da solução entende-se a codificação do cromossomo adotada no contexto do AG, a qual é discutida com detalhes na seção 3.4 páginas 56, não devendo, portanto, ser confundida com a representação do problema de sequenciamento apresentada na seção 2.3 páginas 35.

Outro ponto interessante é que as formas de representar as soluções no AG, quando aplicadas ao processo de busca, podem gerar soluções que não sejam aplicáveis à realidade industrial, podendo tornar-se uma sequência não factível, não executável. De fato, uma representação inadequada pode levar o AG a uma busca superficial ou na geração de novos indivíduos que resultem em soluções também não factíveis.

Esses efeitos são relativamente bem difundidos nas representações mais usuais como a baseada em listas de números inteiros, a qual forma um cromossomo no AG com números inteiros, no problema real, representa a lista de sequência em determinadas máquinas.

Quando se utiliza representações com cromossomos binários, incomum para problemas de sequenciamento, como é o caso da representação baseada no conceito de semente dinâmica proposta recentemente (GRASSI, 2014), faz-se necessário uma análise do desempenho relativo à geração de soluções não factíveis. Entende-se como semente dinâmica uma matriz baseada na sequência de tarefas em cada máquina, sempre ordenada pela M1 em diante, esta é a primeira solução que será usada pelo AG, o qual iniciará o processo de busca, as outras soluções geradas a partir desta semente serão soluções já otimizadas.

Cumpre ainda destacar que a representação por semente dinâmica é dita indireta, pois é necessária ainda outra etapa de conversão do cromossomo, para se chegar a sequência da solução que se aplica ao problema real.

1.1 FORMULAÇÃO DO PROBLEMA

O problema dessa pesquisa pode ser formulado na seguinte questão:

> Qual a melhor Representação cromossômica da solução nos AGs em problemas de sequenciamento da produção em *job shop?*

1.2 OBJETIVOS

Para uma melhor compreensão, os objetivos foram divididos em: objetivo geral e objetivos específicos.

1.2.1 Objetivo Geral

Realizar um estudo comparativo de diferentes representações cromossômicas da solução no AG em problemas de sequenciamento da produção em *job shop*, considerando duas representações com base em números inteiros e uma representação binária.

1.2.2 Objetivos Específicos

1. Identificar, com base na pesquisa bibliográfica, os tipos de operadores genéticos de cruzamento para representações binárias indiretas.
2. Determinar uma configuração adequada para o AG com representação binária indireta com semente dinâmica, no que diz respeito aos operadores de cruzamento.
3. Investigar uma possível correlação entre a proporção de soluções não factíveis geradas pela representação binária com semente dinâmica, quando se altera os operadores de cruzamento e também as taxas de cruzamento.
4. Comparar a forma de representação binária indireta utilizando semente dinâmica com representações utilizando lista de números inteiros, segundo o critério da proporção de soluções não factíveis geradas.
5. Comparar as representações em relação ao custo computacional e ao valor de *makespan* da solução encontrada.

1.3 DELIMITAÇÃO DO ESTUDO

O presente trabalho de pesquisa está focado nas representações das soluções baseadas por listas de números inteiros (lista de prioridades e lista de

operações) e na representação binária indireta com semente dinâmica proposta por Grassi, (2014). Outras formas de representação como, por exemplo, a representação por chaves aleatórias (BEAN, 1994; GONÇALVES; RESENDE; RUIZ et al., 2015), que usam uma codificação real não foram investigadas.

Os experimentos para a obtenção dos resultados foram realizados considerando os problemas LA01 até LA20, do grupo de exemplares de problemas de sequenciamento da produção em *job shop* (do inglês, *JSSP*) definidas por Lawrence (1984), as análises feitas são restritas a esse conjunto de problemas.

1.4 MOTIVAÇÃO E JUSTIFICATIVAS DA DISSERTAÇÃO

A literatura especializada mostra que as representações da solução no AG produzem resultados diferentes para o problema de sequenciamento em *job shop* (ABDELMAGUID, 2010). Teoricamente a proporção de soluções não factíveis geradas pela representação exerce uma influência no desempenho do AG, no que tange encontrar uma solução de melhor qualidade em um menor tempo computacional.

Por outro lado, representações que não produzem soluções não factíveis geram soluções redundantes que também, podem atrapalhar o desempenho do método de otimização afetando também no custo computacional, esse é o caso da representação baseada em lista de operações.

1.5 ESTRUTURA DO TRABALHO

A continuação deste trabalho está organizada da seguinte forma: o capítulo 2 trata das características do sequenciamento da produção, com os conceitos e noções de sequenciamento da produção, ambientes de produção, problema de sequenciamento *job shop*. Nesse capítulo ainda são fornecidas

noções básicas sobre complexidade computacional e a necessidade de algoritmos metaheurísticos na busca de soluções com tempo computacional adequado.

O capítulo 3 é dedicado ao assunto sobre AG, correlacionando os conceitos básicos da genética com o algoritmo, descreve os operadores genéticos de cruzamento, seleção e mutação, bem como as representações da solução para problemas *job shop*, destacando a apresentação sobre codificação e sua relação com as várias representações, diretas ou indiretas, abordadas.

O capítulo 4 trata sobre Materiais e Métodos, discorre sobre as características da pesquisa, parâmetros adotados, bibliotecas de algoritmos genéticos, família de problemas testados e condições de possíveis travamentos gerando soluções não factíveis.

O capítulo 5 mostra os diversos resultados dos vários experimentos. Utilizando tabelas, gráficos e estatísticas para as comparações, este capítulo foi sequenciado de maneira que torne o entendimento para o leitor fácil e possa correlacionar os experimentos. São apresentadas na sequência dos experimentos as análises correspondentes.

O capítulo 6, as conclusões e limitações bem como sugestões de melhorias do algoritmo e sugestões para pesquisas futuras.

2 SEQUENCIAMENTO DA PRODUÇÃO EM JOB SHOP: REPRESENTAÇÕES E COMPLEXIDADE DA SOLUÇÃO

Este capítulo apresenta e discute os conceitos fundamentais para o entendimento do trabalho. Inicia-se apresentando as classificações dos ambientes de produção e dos problemas de sequenciamento nesses ambientes, com especial destaque para o *job shop*. Discute brevemente a complexidade computacional desse problema e, finalmente, as técnicas usadas frequentemente em sua solução.

2.1 O PROBLEMA DO SEQUENCIAMENTO DE PRODUÇÃO

O sequenciamento da produção pode ser entendido como o processo de atribuição de um ou mais recursos para a execução de certas atividades as quais, em sua execução, vão exigir certa quantidade de tempo (LUKASZEWICZ, 2005). Os recursos podem ser representados como máquinas e as atividades, que serão processadas em cada máquina, de tarefas, necessárias para a fabricação do produto. Cada tarefa em cada máquina será chamada de *job,* que representará o conjunto de uma ou mais tarefas. O sequenciamento da produção é um fator preponderante, pois visa reduzir o tempo total exigido na execução de um conjunto de tarefas, ou como já foi dito *job*.

O problema de sequenciamento da produção, consiste em encontrar melhores combinações do sequenciamento de tal forma a otimizar a utilização das máquinas e consequentemente reduzir os tempos e custos da produção. Para se melhorar o sequenciamento é necessário primeiramente conhecer bem o ambiente em questão, suas limitações e também o objetivo a ser minimizado, sendo assim, dependendo do ramo da indústria, do tipo do produto e das características inerentes a produção a classificação do sequenciamento será diferente (PINEDO, 2009).

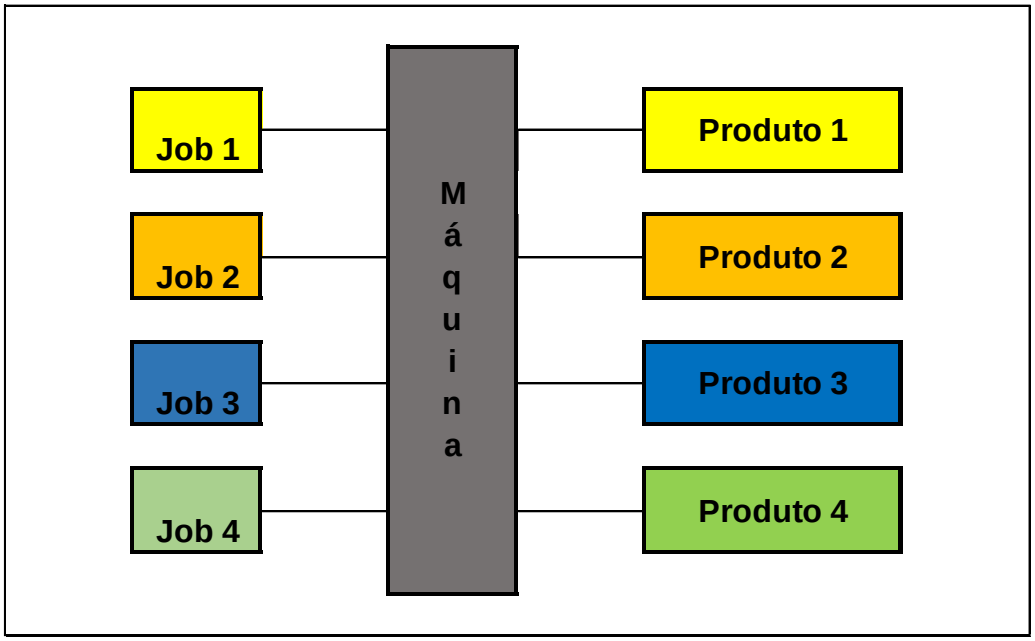
A classificação e a dificuldade da solução dos problemas de sequenciamento estão intimamente ligadas aos diferentes tipos de ambientes de produção e demais características (ARENALES et al, 2007), como mencionado anteriormente.

Segundo Pinedo (2012), os problemas de sequenciamento da produção podem ser classificados pelos parâmetros α | β | γ: α representa o ambiente da máquina, β as características de processamento e suas restrições, e por último, γ representa o objetivo a ser minimizado.

2.1.1 Ambiente da máquina – parâmetro α

As possíveis classes de problemas de sequenciamento especificadas pelo parâmetro α são: máquina única, máquinas idênticas em paralelo, máquinas em paralelo com diferentes velocidades, máquinas distintas em paralelo, *flow shop, flow shop* flexível, *job shop*, *job shop* flexível e *open shop.* Cada um dos ambientes citados é respectivamente descrito.

Figura 1 – Exemplo de Máquina Única

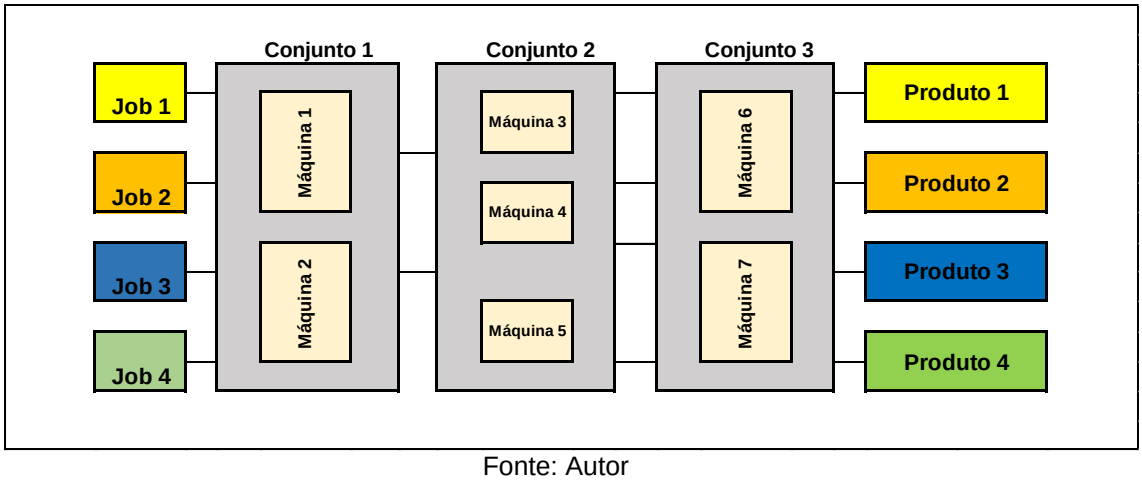


Fonte: Autor

a) Máquina Única (*Single machine*). O ambiente baseado em uma única máquina é o modelo mais simples e o mesmo serve como referência para se estudar heurísticas, que poderão ser aplicadas a modelos mais complexos, pois na prática os modelos mais complexos são decompostos em subproblemas baseados em máquinas únicas (PINEDO, 2012). Neste ambiente, todos os *job*s e suas respectivas tarefas são processados em uma única máquina especializada. Esse tipo de ambiente é ilustrado na Figura 1.

b) Máquinas idênticas em paralelo (*Identical machines in parallel*). O ambiente baseado em máquinas idênticas em paralelo, consiste na formação de um ou mais conjuntos de máquinas idênticas executando a mesma função. Entende-se que possuem as mesmas velocidades, onde os *jobs* exigem uma operação única e podem ser processadas em qualquer uma das máquinas daquele conjunto. Este ambiente é ilustrado na Figura 2.

Figura 2 – Exemplo de máquinas idênticas em paralelo



Fonte: Autor

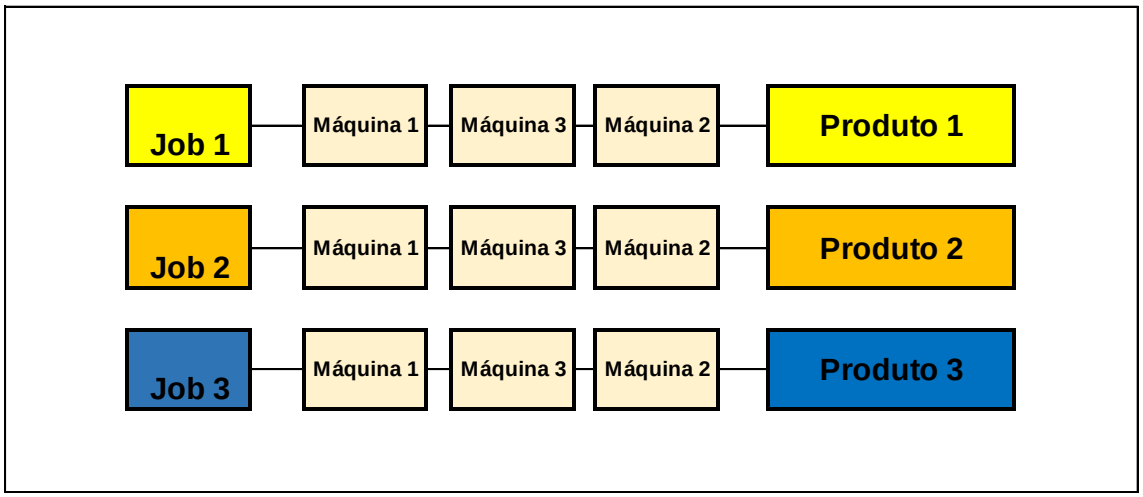
c) Máquinas em paralelo com diferentes velocidades (*Machines in parallel with different speeds*). O ambiente baseado em máquinas em paralelo, mas com diferentes velocidades, diferencia do ambiente anterior, pois há conjuntos de máquinas que executam a mesma função, mas entre as máquinas não há uniformidade de velocidade. Pode-se dizer que ocorrerá o caso de se ter máquinas mais rápidas que outras para determinada tarefa. Este ambiente

também é conhecido como máquinas paralelas uniformes, sendo que o modelo anterior está contido neste modelo, como subproduto. Na Figura 2 as máquinas do mesmo conjunto possuem velocidades iguais, já neste modelo, dentro dos conjuntos as máquinas possuem velocidades diferentes.

d) Máquinas distintas em paralelo (*Unrelated machines in parallel*). Os ambientes formados por máquinas distintas em paralelo são definidos como máquinas em paralelo que executam a mesma função, mas possuem diferentes recursos ou capacidades, ou seja, os tempos de processamento dos *jobs* dependem da máquina a que estão atribuídos. O desafio deste ambiente é a escolha da máquina mais apropriada para cada *job* e, também, em definir a melhor sequência para cada *job* (LÓPEZ et al., 1995; MOKOTOFF, 2001; PFUND et al., 2004; LIN et al., 2011). Neste ambiente cada *job* deve ser processado exatamente uma vez em cada máquina; cada *job* possui um tempo de processamento que depende da máquina na qual será alocada e existem tempos de preparação entre as tarefas.

e) *Flow shop*. Neste ambiente existem *m* máquinas em série. Cada *job* para ser concluído, deve passar em todas as *m* máquinas e seguir exatamente o mesmo percurso, ou seja, cada *job* têm que obedecer a mesma sequência pré-estabelecida. Após a conclusão da tarefa em determinada máquina, aguarda-se em fila, para posteriormente ser processado e continuar a sequência. As filas trabalham na disciplina *First In First Out* (FIFO), significa que um *job* não tem precedência em relação a outro. O exemplo de um *flow shop* é ilustrado na Figura 3.

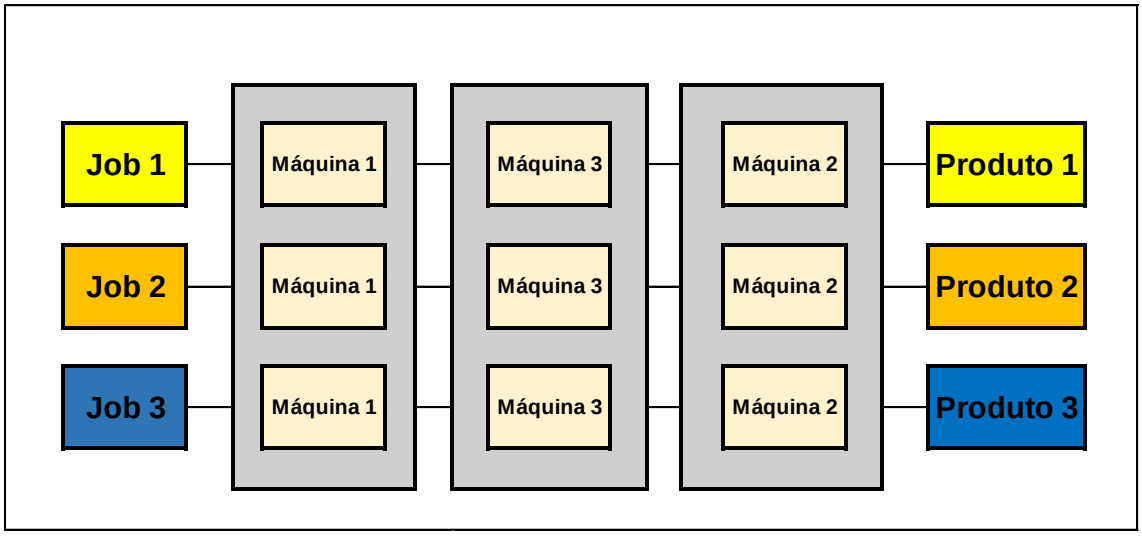
Figura 3 – Exemplo *flow shop*



Fonte: Autor

f) *Flow shop* flexível. O ambiente *flow shop* flexível é uma generalização do *flow shop* agregando-se o ambiente de máquinas paralelas. Há vários estágios em série, sendo que cada estágio é formado por um conjunto de máquinas idênticas em paralelo. Cada *job* deverá seguir exatamente a mesma sequência pré-estabelecida para se visitar os estágios. Cada *job* pode ser processado em qualquer máquina do conjunto daquele estágio apenas uma vez, as filas entre os vários estágios podem ou não funcionarem de acordo com a disciplina FIFO. Este ambiente é mostrado na Figura 4.

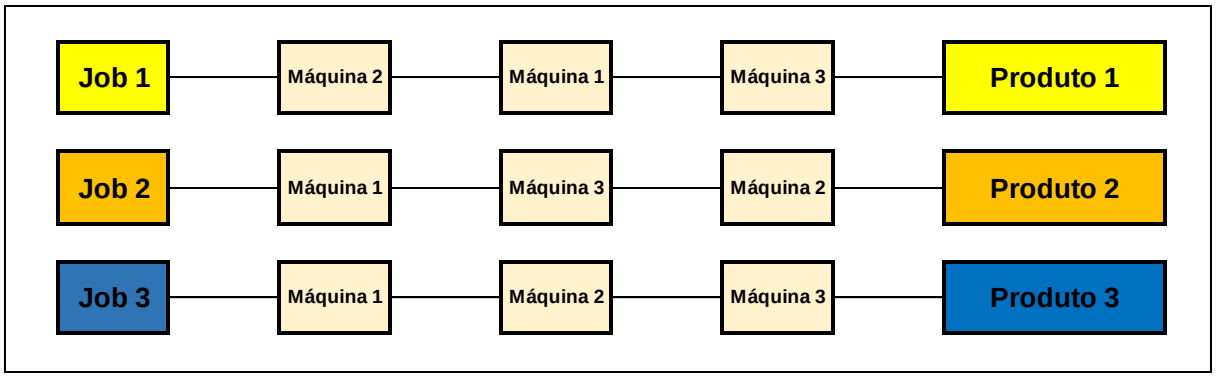
Figura 4 – Exemplo *flow shop* flexível



Fonte: Autor

g) *Job shop*. Num ambiente tipo *job shop*, existem *m* máquinas que não estão em série e cada *job* tem sua própria rota pré-determinada a ser seguida, sendo única, diferente das rotas dos outros *job*s. Cada *job* visita cada máquina, no máximo, uma única vez, ou seja, poderá existir máquinas que não serão visitadas por determinados *job*s. O problema de sequenciamento *job shop* ou como conhecido na língua inglesa, *job shop schedunlig problems* (JSSP) é o objetivo deste estudo, sendo assim será tratado detalhadamente em sua devida seção. Um exemplo de *job shop* é apresentado na Figura 5.

Figura 5 – Exemplo de *job shop*



Fonte: Autor

h) *Job shop* flexível. Semelhante ao que acontece com o *flow shop* e o *flow shop* flexível, o *job shop* flexível também é uma generalização do *job shop* com o ambiente de máquinas paralelas. Temos várias fases em série, sendo que cada fase é formada por um conjunto de máquinas idênticas em paralelo. Cada *job* tem sua própria rota pré-determinada a qual deverá ser seguida, e é única. Lembrando que o *job* deverá ser tratado em apenas uma máquina do conjunto e qualquer máquina daquele conjunto pode atendê-lo

i) *Open shop*. No ambiente open shop cada job será processado em cada uma das máquinas. No entanto, alguns destes tempos de processamento podem ser zero. Não há restrições com relação ao encaminhamento de cada job através do ambiente de máquina. O programador está autorizado a determinar uma rota para cada *job* e jobs diferentes podem ter diferentes rotas. O problema

de sequenciamento open shop é semelhante ao problema de sequenciamento job shop (JSSP), com a ressalva de que não há uma ordenação pré-estabelecida no open shop sobre as tarefas dentro de qualquer *job.*

2.1.2 Características de processamento e suas restrições - parâmetro β

As possíveis variáveis do parâmetro β apresentadas por Pinedo (2012), são as características de processamento e suas restrições que serão descritas a seguir:

a) Data de liberação (*Release dates*). A data de liberação é a data inicial para se começar a processar determinado *job*, significa que o *job* não pode ser processado antes desse tempo. Quando o *release date* não é definido, significa que o *job* pode ser processado em qualquer tempo.

b) Preempção (*Preemptions*). A preempção ou preferência, implica que o *job* com maior prioridade é processado imediatamente, ou com certa prioridade escolhida pelo programador, interrompendo o processamento do *job* com menor prioridade. Ao terminar, o *job* de menor prioridade volta a ser processado, podendo continuar o processo de onde parou ou então reiniciá-lo.

c) Restrições de precedência (*Precedence constraints*). As restrições de precedência basicamente consistem em que um ou mais *job*s precisam ser concluídos, para que outro *job* seja autorizado a iniciar o seu processamento.

d) Sequência que depende de tempo de preparação (*Sequence dependent setup times*). Há sequências para determinados *jobs* que necessitam de uma nova configuração em cada máquina. Esta configuração exige um tempo de preparação, este tempo é considerado como uma das restrições, pois o tempo de preparação entre dois *jobs*, depende da máquina escolhida e também das características dos próprios *jobs* que serão processados, neste interim as máquinas não podem processar nenhum *job*.

e) Famílias de *job*s (*job families*). Agrupa-se *Jobs* em famílias quando há muita semelhança entre eles. *Jobs* da mesma família podem ter diferentes tempos de processamento, porém podem ser processados numa mesma máquina e em sequência, sem necessidade de qualquer configuração. Essa prática promove um ganho no sequenciamento, pois não existirá o tempo de *setup* das máquinas.

f) Processamento em lote (*Batch processing*). O processamento em lote é muito comum em linhas de produção, neste caso considera-se que os tempos de processamento de um lote podem não ser exatamente iguais e todo o lote somente é concluído quando o último *job* foi concluído, o que implica que o tempo de conclusão de todo o lote é determinado pelo *job* com o tempo de processamento maior.

g) Quebra de máquinas (*Breakdowns*). A indisponibilidade de uma ou mais máquinas é uma realidade nos problemas de sequenciamento da produção. O tempo em que as máquinas estão aguardando, a manutenção e o próprio tempo de manutenção acabam afetando toda a produção, pois outras máquinas deverão assumir as tarefas, podendo gerar um tempo de fila maior que o esperado. Sendo assim, esta restrição deve ser muito bem ponderada, devendo ser considerada mesmo quando as manutenções são programadas.

h) Restrições relativas à máquina (*Machine eligibility restrictions*). Essas restrições se aplicam quando existem *m* máquinas em paralelo e determinada máquina daquele conjunto tem alguma restrição relativa a um *job* especifico.

i) Permutação (*Permutation*). No ambiente *flow shop* a ordem de sequenciamento 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. Isto implica que a ordem de execução dos *jobs* na primeira máquina é igual na próxima e assim sucessivamente até a última durante todo o processo.

j) Bloqueio (*Blocking*). O bloqueio de uma ou mais máquinas, pode ocorrer por vários motivos em ambientes flow shop por exemplo, quando existem locais de armazenamento de tamanho limitado entre duas máquinas consecutivas, impedindo a continuação do processamento de determinado job e afetando toda a cadeia.

k) Sem espera (*No-wait*). Este evento ocorre em ambientes onde não existem filas intermediárias entre duas máquinas, para *job*s que não podem esperar pois prejudicariam a integridade de determinado produto, exemplo que ocorre na linha da indústria metalmecânica com produtos que devem operar em dadas temperaturas e não podem resfriar.

l) Recirculação (*Recirculation*). Consiste num job visitar mais de uma vez a mesma máquina ou um conjunto de máquinas paralelas.

2.1.3 Objetivo a ser minimizado – parâmetro γ

O parâmetro γ está relacionado com o objetivo que se deseja otimizar. Há itens de medição que são comumente utilizados nos processos de produção, estes itens compõem as funções objetivo. Na sequência serão apresentados estes itens para uma melhor compreensão do parâmetro γ.

a) Tempo de término de um *job* (*Completion time* - C_j). Corresponde ao instante em que termina o processamento do *job* j.

b) Tempo de fluxo de um *job* (*flowtime* - F_j). Corresponde à soma dos tempos de espera e de processamento de um *job*.

c) Desvio do tempo de término de um *job* (*Lateness* - L_j). Definido matematicamente como a diferença entre o tempo de término de um *job* e a data de entrega do *job* (dj), definida no processo. $L_j = C_j - d_j$, caso o valor de Lj seja

positivo significa que o processo de entrega do *job* está atrasado, caso contrário significa que o processo de entrega do *job* está adiantado.

d) Atrasados (*Tardiness* – T_j). A definição é semelhante ao parâmetro *lateness*, com a única diferença, não se considerar os *job*s que estiverem adiantados. Entende-se como o tempo de atraso dos *job*s. Tj = max (L_j, 0).

e) Antecipados (Earliness - Ej). A definição também é semelhante ao lateness, mas neste caso não se considera os jobs que estiverem atrasados. Entende-se como o tempo de antecipação dos jobs.

Os critérios de otimização também podem estar relacionados com determinados objetivos, por exemplo, o tempo de conclusão das tarefas, o tempo de fluxo, ou ainda nas datas de entrega, os principais critérios de otimização para problemas de sequenciamento da produção são descritos a seguir (PINEDO, 2012; RODAMMER; WHITE, 1988).

a) *Makespan* ($C_{máx}$). Também conhecido como tempo total de processamento, é equivalente ao tempo de conclusão do último *job* a deixar o sistema. Quando o *makespan* é pequeno implica geralmente numa boa utilização das máquinas daquela formação.

b) Tempo total de término de um *job* (*total completion time*). A somatória do tempo de término de todos os *jobs*.

c) Tempo total de término ponderado (*total weighted completion time*). Representa a soma ponderada dos instantes de término de cada *job*, de acordo com um peso atribuído.

d) Tempo do fluxo total dos *job*s (*total flowtime*). A somatória de todos os tempos de fluxo.

e) Soma ponderada dos tempos de fluxo (*Total weighted flowtime*). A soma ponderada de todos os tempos de fluxo, de acordo com o peso atribuído.

f) Desvio máximo em relação ao tempo do término dos *job*s (*Maximum lateness*) ($L_{máx}$).

g) Soma dos atrasos (*Total tardiness*).

h) Soma ponderada dos atrasos (*Total weighted tardiness*).

i) Número total de *job*s em atraso (*Number of tardy jobs*).

j) Soma dos atrasos e antecipações (*Total earliness-tardiness penalty*).

k) Soma ponderada dos atrasos e antecipações (*Total weighted earliness-tardiness penalty*)

l) Minimização do tempo de preparação das máquinas.

m) Minimização do custo de preparação das máquinas.

2.2 PROBLEMAS DE SEQUENCIAMENTO *JOB SHOP*

O ambiente *Job shop* tem sido muito utilizado na produção industrial nos últimos anos. Com a sofisticação tecnológica e o aumento dos recursos computacionais, diversas áreas da indústria melhoraram a sua produtividade neste tipo de ambiente. O número de pesquisas sobre o assunto tem aumentado em alguns setores como: fábricas de vidro (ALVAREZ-VALDES et al., 2005), indústria têxtil e indústria do plástico (PAN, 2012), indústria química (LI; PAN, 2012), indústria automotiva e afins (SELS; STEEN; VANHOUCKE, 2011), entre outras.

O problema de sequenciamento *job shop* (JSSP), é classificado de acordo com os critérios: forma de atendimento da demanda, processo de chegada das ordens de fabricação e distribuição do tempo de processamento dos *jobs* (HAX; CANDEA, 1984; PINEDO, 2012).

O critério da forma de atendimento da demanda segundo a classificação dada por Hax e Candea (1984), pode ser classificado em *job shop* aberto, fechado ou misto. Para o *job shop* aberto a ordem de fabricação dos produtos é sob encomenda, os produtos são fabricados de acordo com os pedidos de clientes. Para o *job shop* fechado a ordem de fabricação é conhecida, suas demandas podem ser previstas, trabalhando-se assim em lotes maiores e com possibilidade de estocagem do excesso produzido. Para o *job shop* misto tem-se uma mistura dos ambientes aberto e fechado.

O critério processo de chegada das ordens de fabricação, segundo a classificação dada por Pinedo (2012), classifica-se em estático ou dinâmico. Estático quando as ordens a serem programadas estão todas disponíveis no início da programação e dinâmico quando as ordens vão chegando ao longo do tempo. Neste caso, pode-se verificar chegadas determinísticas, quando os instantes de chegada são previamente conhecidos, ou chegadas estocásticas, quando os instantes de chegada seguem uma distribuição de probabilidade.

Referente à distribuição do tempo de processamento dos *job*s (Pinedo, 2012), classificam-se em determinísticos, quando os tempos de processamento dos *job*s são fixos e conhecidos, ou estocásticos, quando os tempos de processamento dos *job*s são representados por uma distribuição de probabilidade, com variáveis aleatórias. Os critérios apresentados acima para a classificação estão reunidos no Quadro 1.

Num sequenciamento tipo *job shop*, cada ordem é única com rotas pré-estabelecidas, diferentes e processadas pelo menos uma vez em cada uma das máquinas, sendo que os problemas dentro deste ambiente são conhecidos como problemas de sequenciamento *job shop* (JSSP).

As características e o comportamento do JSSP envolvem o processo de tomada de decisão, objetivando encontrar uma sequência de tarefas que minimize, por exemplo, o tempo de entrega, ou maximize a utilização das máquinas aumentando a capacidade produtiva.

Quadro 1 – Resumo da classificação de ambientes *job shop*

Critério	Classificação	Descrição
Forma de atendimento da demanda	*job shops* abertos	Fabricação sob encomenda
	job shops fechados	Fabricação fixa
	job shops mistos	Fabricação composta pela mistura dos ambientes aberto e fechado
Processo de chegada das ordens de fabricação	*job shops* estáticos	Todas as ordens estão disponíveis no início
	job shops dinâmicos	As chegadas das ordens seguem uma distribuição de probabilidade, chegam ao longo do tempo
Distribuição do tempo de processamento dos *Jobs*	*job shops* determinísticos	Tempos de processamento são fixos e conhecidos
	job shops estocásticos	Tempos de processamento são representados por uma distribuição de probabilidade.

Fonte: Pinedo (2012); Hax, Candeia (1984)

Nos diferentes cenários do ambiente de produção (FAN; ZHANG, 2010) pode-se classificar as principais características e limitações do JSSP:

Todas as máquinas estão disponíveis no tempo 0 e todos os *jobs* são liberados no tempo 0.

- Cada *job* pode ser processado por apenas uma máquina de cada vez.
- Processos simultâneos com a mesma máquina não são permitidos.
- Cada *job* é composto por várias tarefas de um processo.
- Cada *job* pode visitar uma máquina apenas uma vez.
- Cada tarefa deve ser executada em máquinas específicas.
- A sequência de operações para cada *job* sempre está pré-definida e não pode ser modificada.
- O tempo de processamento de todas as operações é conhecido.

- Não há precedência às limitações entre as operações de diferentes *jobs.*
- Cada operação do início ao fim não pode ser interrompida por outros processos.

2.3 FORMAS DE REPRESENTAÇÃO DE PROBLEMAS *JOB* SHOP

As formas de representar problemas de sequenciamento organizam a disposição das informações melhorando a visão geral de cada problema *job shop*. O problema *job shop* é representado pelo mnemônico $n/m/G/C_{max}$, onde *n* representa a quantidade de *job*, *m* a quantidade de máquinas, *G* significa que é da família *job shop* e C_{max} é o objetivo a ser otimizado, por exemplo, o *makespan* (Pinedo 2012).

Dentre as várias formas de se representar os problemas de sequenciamento *job shop* destacam-se a sequência tecnológica, matrizes, gráficos disjuntivos, gráficos de Gantt, descritos a seguir:

- A representação por sequência tecnológica é bem versátil, pois apresenta a sequência que cada *job* deve seguir em cada máquina e o tempo de processamento de cada *job* nas máquinas, como visualizado na Tabela 1.

Tabela 1 – Sequência Tecnológica do JSSP

Job	**Máquina (Tempo de processamento)**		
1	1(2)	2(1)	3(1)
2	3(1)	2(1)	1(1)
3	2(1)	1(2)	3(2)

Fonte: Autor

- A representação por matrizes das operações e tempos de processamento é gerada a partir da sequência tecnológica. A matriz O_{jm},

representa a operação do *job j* na máquina *m* e a matriz P_{jm} representa o tempo de processamento do job j na máquina *m*, como visualizado na Figura 6.

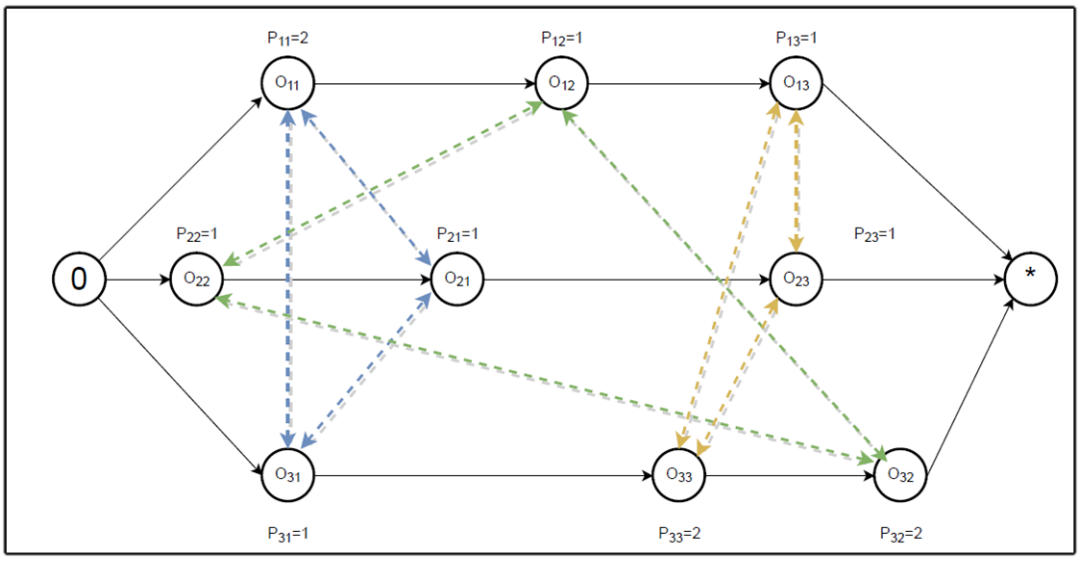
Figura 6 – Matrizes das operações e dos tempos de processamentos

$$Ojm = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix} \qquad Pjm = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 2 \end{bmatrix}$$

Fonte: Autor

- Representação por gráficos disjuntos: a representação por gráficos disjuntos mostra o sequenciamento visualmente, ficando fácil a interpretação. Mostra as operações dentro de cada nó representadas por O_{jm}, a qual indica a execução do *job* j na máquina m e, externamente, o tempo de processamento do *job* j na máquina m, representado por Pjm. As orientações das setas tracejadas dos arcos disjuntos mostram as possíveis prioridades das sequências de operações distintas na mesma máquina. As setas contínuas e os arcos conjuntos representam a ordem da sequência tecnológica de cada job com origem no nó zero e término no nó asterisco, como visualizado na Figura 7.

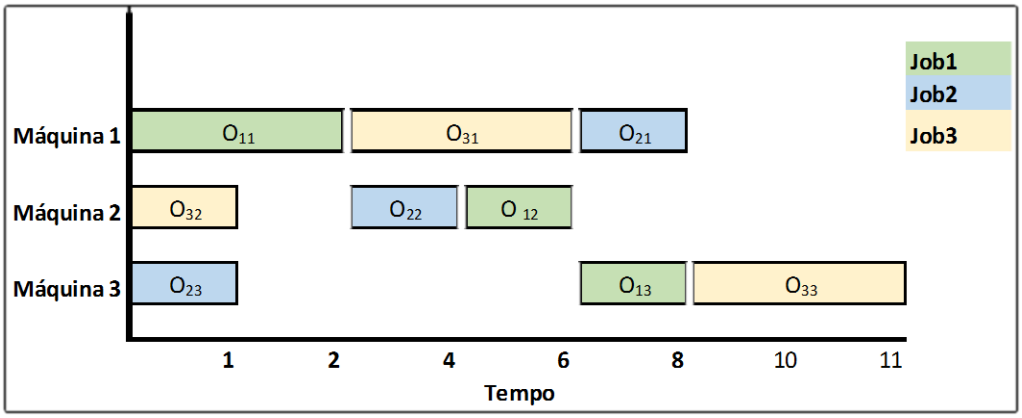
Figura 7 – Gráfico disjunto



Fonte: Autor

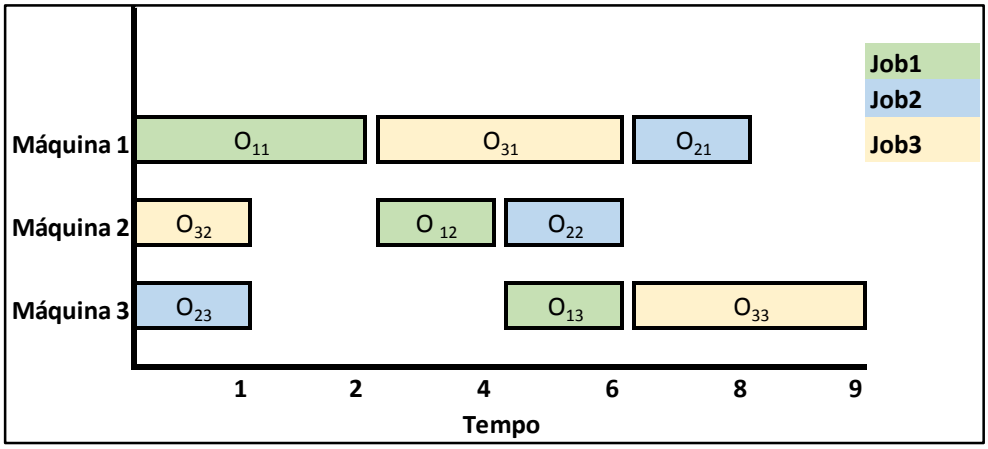
- Representação por diagrama de Gantt: a representação por diagrama de Gantt é uma representação visual muito fácil de compreender, sendo muito utilizada para os problemas de sequenciamento. O gráfico de Gantt tem sido utilizado para o controle da produção, por apresentar as operações bem como o tempo gasto para executá-las. A sua facilidade consiste em representar em linha, problemas n x m, n *job*s e m máquinas, mostrando a sequência completa das operações dos *job*s em determinada máquina. Na Figura 8 é visualizado o *job shop* definido com os dados da Tabela 1, representando cada operação por $O_{jk,}$ com $1 \leq j \leq n$ e $1 \leq k \leq m$.

Figura 8 – Diagrama de Gantt para 3 máquinas e 3 *jobs*



Fonte: Autor

Figura 9 – Diagrama de Gantt para 3 máquinas e 3 *jobs makespan* menor



Fonte: Autor

Na Figura 9 é possível visualizar a mesma sequência tecnológica, mas com uma combinação diferente, possibilitando um *makespan* menor.

Para subsidiar a discussão sobre a escolha do método de otimização para problemas de sequenciamento da produção em *job shop*, iniciada na seção 2.1, pg. 22, apresenta-se na sequência alguns conceitos relativos à complexidade computacional relativos a essa classe de problemas.

2.4 COMPLEXIDADE COMPUTACIONAL

A teoria da complexidade computacional é um ramo da matemática que procura classificar os problemas computacionais de acordo com o seu grau de dificuldade, objetivando classificar os problemas que podem ser resolvidos por algoritmos computacionais de forma eficiente em diversas classes (OLIVEIRA, 2010). Essas classes são divididas de acordo com a quantidade de recursos computacionais necessários e suficientes para resolver cada problema. Para medi-las podem ser abordadas duas instâncias: espacial e temporal.

Na complexidade espacial preocupa-se com a quantidade de recursos físicos. No caso *hardware* do computador, como memória, disco e processador, esses recursos influenciam no desempenho da execução do algoritmo. Já na complexidade temporal preocupa-se com o tempo de execução. Os pesquisadores reconhecem que nem todos os problemas podem ser resolvidos tão rapidamente, são classificados como problemas computacionalmente difíceis de resolver (MITCHELL, 1997; TEIXEIRA, 1998).

Na teoria da complexidade computacional, os problemas são classificados de acordo com o tempo gasto pelo algoritmo para encontrar a solução, nesta classificação utilizam-se as letras P e NP para denotar classes de problemas computacionais. P significa “tempo determinístico polinomial” e NP “tempo não determinístico polinomial”. (OLIVEIRA, 2010; JIAN-MING, 2013; ERICKSON, 2009).

Os problemas P, são conjuntos de problemas com soluções que podem ser encontradas de forma eficiente e em tempo viável pois, podem ser resolvidos em

tempo polinomial por algum algoritmo determinístico (SHPARLINSKI, 2003). Os problemas NP, são conjuntos que contém os problemas computacionais com soluções que podem ser verificadas de forma eficiente, mas não podem ser encontradas eficientemente. Significam e indicam problemas para os quais se utiliza algoritmos não determinísticos, ou seja, algoritmos que geram soluções factíveis cuja viabilidade pode ser verificada em tempo polinomial. Ainda dentro dos problemas NP, existem os chamados NP-hard, problemas NP muito difíceis de se resolver (SHPARLINSKI, 2003).

Cobham e Edmonds (1965) entre outros (EISELT; SANDBLOM, 2004), sugerem que um algoritmo é eficiente quando a função do tempo de execução definida pelo número de passos de sua rotina é limitada por uma função algébrica, ou seja, por uma função polinomial.

Conforme a equação (1), um polinômio de grau 3 é composto pela soma de vários monômios, neste caso cada monômio representando o tempo de execução de determinados passos dentro do algoritmo, sendo assim, mesmo que as quantidades de passos aumentem, resultará no máximo um polinômio de grau maior, ou com maior quantidade de parcelas, resultando numa soma do tempo no final, conforme a equação (2).

$$P(x) = x^3 + 3x^2 + 3x + 1 \;\rightarrow\; (x+1)^3 = \sum_{k=0}^{3} \binom{3}{k} x^k 1^{3-k} \qquad (1)$$

$$P(x) = x^4 + 8x^3 + 24x^2 + 32x + 16 \;\rightarrow\; (x+2)^4 = \sum_{k=0}^{4} \binom{4}{k} x^k 2^{4-k} \qquad (2)$$

O tempo neste caso cresce sempre somando parcelas na função polinomial, mas nunca se multiplicando. Esta definição apresenta diversas propriedades interessantes, tem sido muito utilizada e tornou-se amplamente aceita afirmando que algoritmos polinomiais são computacionalmente viáveis.

$$F(x) = (4)^x + (7)^{2x} \quad (3)$$

Nos problemas classificados como NP, o tempo para resolução dos passos não cresce em tempo polinomial, mas sim, de forma exponencial. Cada novo passo multiplica o tempo da execução do algoritmo. Para problemas com muitos passos o tempo resultante pode ser muito alto, tornando a espera pelo processamento impraticável. Um exemplo de função não polinomial é mostrado na equação (3).

Um exemplo comparativo do custo computacional é mostrado na Tabela 2, onde são avaliadas funções polinomiais e não polinomiais. Segundo as pesquisas de Devlin (2008), N representa o número de passos da rotina, em questão. Considera-se que os tempos são obtidos analisando um computador que consiga realizar um milhão de operações aritméticas básicas por segundo.

Tabela 2 – Crescimento em tempo exponencial

Função tempo/ complexidade	**Quantidade de Dados: N**				
	10	20	30	40	50
N	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s
N^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0036s
N^3	0,001s	0,008s	0,027s	0,064s	0,125s
2^N	0,001s	1,0s	17,19 min	12,7 dias	35,7 anos
3^N	0,059s	58 min	6,5 anos	3.855 séculos	200.000.000 séculos

Fonte: Devlin, 2008

Garey e Johnson (1976) apresentam a seguinte classificação para os problemas, segundo sua complexidade:

a) **Indecidível:** são problemas tão difíceis, que nenhum algoritmo pode ser utilizado para resolvê-los. Há inúmeros problemas considerados desta classe, tais como o problema de decisão de Hilbert, o problema que envolve o teorema de Rice, o problema de parada da máquina de Turing, o problema da correspondência de Post, entre outros. Para se ter uma melhor visão há abrangente literatura a respeito (BROOKSHEAR, 2012; HALAVA, 1997; RECHARD et al., 2015; BAETEN; LUTTIK; VAN TILBURG, 2013; JEZ; OKHOTIN, 2014; ALHAZOV et al., 2013).

b) **Intratável:** problemas decidíveis, porém difíceis para os quais possivelmente não existe algoritmo que os resolvam em tempo polinomial. Entre os mais conhecidos estão os problemas de sequenciamento da produção, do caixeiro viajante, de partição em triângulos, de circuito hamiltoniano, de roteamento de arcos entre outros. Há abrangente literatura a respeito (HIRATA; YAMADA; HARAO, 2004; BOSWELL, 2009; SHERAFAT, 2004; LAKSHMIPATHY; WINKLMANN, 1986; JIN; LIANG, 2014).

c) **Tratável:** problemas para os quais existe algoritmo que os resolvam em tempo polinomial. Entre os mais conhecidos pode-se citar o problema dos circuitos eulerianos, o problema de cobertura de borda (*edge cover*), problema de gráficos com duas colorações, cardinalidade equivalente (*cardinality matching*), entre outros. (SHERAFAT, 2004; LAKSHMIPATHY; WINKLMANN, 1986; AZAD; BULUÇ; POTHEN, 2015; FANG et al., 2014).

O problema de sequenciamento da produção em *job shop* (JSSP), é caracterizado como um problema de otimização combinatória pertencente à classe NP-hard (FAN; ZHANG, 2010). Segundo Pinedo (2012), problemas de otimização combinatória são aqueles que buscam determinar dentre um subconjunto de possibilidades de soluções aquela que apresenta o menor custo.

No caso do JSSP, isso significa identificar, dentre as várias combinações de escalonamento dos *job*s nas máquinas, a sequência que apresenta menor tempo total de produção.

2.5 MÉTODOS DE OTIMIZAÇÃO PARA JSSP

Em geral, um método de otimização busca identificar, baseado em uma representação do problema a ser otimizado, as melhores soluções possíveis para o problema (soluções sub-ótimas), ou a melhor solução (solução ótima ou ideal). Obviamente o sucesso da técnica utilizada depende de muitos fatores, desde a adequada representação do problema até escolha apropriada dos valores dos parâmetros específicos de cada técnica de otimização (WANG; ZOU, 2003).

Verifica-se que não há regra simples e universalmente válida para determinar o melhor algoritmo para um problema específico (WOLPERT, 1995). Isto considera o fato que um algoritmo deve ser melhor para resolver um determinado problema do que outro, quando parametrizado em função daquele problema, mas tende a não ser tão eficiente quando se tenta generalizá-lo para usá-lo em outro problema (LINDEN, 2012).

O problema JSSP, pela sua complexidade, não pode ser resolvido com um algoritmo de tempo polinomial e é computacionalmente intratável quando se utiliza algoritmos determinísticos (também chamados de algoritmos exatos), como a programação linear (DE ANDRADE, 2009; HILLIER; LIEBERMAN, 2013), por exemplo, que buscam pela solução exata.

Outro caminho viável é a utilização de algoritmos heurísticos e metaheurísticos. Algoritmos heurísticos podem ser definidos como algoritmos inspirados em processos intuitivos que buscam soluções factíveis, não necessariamente as melhores soluções, mas com tempo computacional aceitável. Já os algoritmos metaheurísticos caracterizam-se por liderarem outros algoritmos heurísticos e são muito utilizados na resolução de problemas de otimização. Estes algoritmos não trabalham como os algoritmos exatos, utilizam técnicas que procuram uma boa solução com custo computacional aceitável. Entretanto, não garantem encontrar a solução ótima, mas soluções viáveis e de boa qualidade, pensando em custo computacional.

O uso de algoritmos heurísticos e metaheurísticos permite resolver problemas intratáveis em tempos computacionalmente aceitáveis, fornecendo

soluções ótimas ou muito próximas da otimalidade. Nesse contexto, tempo computacional aceitável significa que uma solução de boa qualidade pode ser encontrada em questão de minutos ou até mesmo segundos, para problemas com até 100 operações, ou seja, o produto entre o número de jobs e o número de máquinas (ABDELMAGUID, 2010).

Assim, diversos são os algoritmos heurísticos e metaheurísticos utilizados para a solução de problemas de JSSP. Algumas dessas técnicas são brevemente descritas no Quadro 2. Em muitos casos, alguns pesquisadores utilizam a combinação de dois ou mais algoritmos, na busca de avanços e melhorias (WANG; LIU, 2013; ZHANG; SONG; WU, 2013; QING-DAO-ER-JI; WANG, 2012; HEINONEN; PETTERSSON, 2007; GONÇALVES; RESENDE, 2015).

Uma das técnicas metaheurísticas mais utilizadas na otimização de problemas de sequenciamento são os Algoritmos Genéticos (AGs), que são algoritmos evolutivos baseados na evolução natural das espécies (REEVES; ROWE, 2003; SIVANANDAM; DEEPA, 2007).

Os Algoritmos Genéticos foram apresentados por John Holland na década de 1960 e desenvolvidos por ele, seus alunos e colegas da Universidade de Michigan entre 1960 e 1970 (DAVIS, 1991; MITCHELL, 1995; MITCHELL, 1998).

Diferente de outras estratégias de evolução e de programação evolucionária, a ideia de Holland não era desenvolver algoritmos para problemas específicos, mas sim estudar o fenômeno de adaptação e com isso, desenvolver maneiras em que os mecanismos de adaptação natural poderiam ser importados para sistemas de computador, mostrando um quadro teórico para a adaptação utilizando o AG (HOLLAND, 1992).

Quadro 2 – Descrição dos algoritmos heurísticos e metaheurísticos

Algoritmo	Descrição	Referências
Busca Tabu	Consiste em trabalhar com a busca local armazenando conhecimento sobre o espaço de busca para que o algoritmo não retorne a uma posição já visitada.	BANNERS, 1996; CHAMBERS, 1996; GLOVER, 1986; GLOVER, TAILLARD, DE WERRA, 1993; DE WERRA, 1989; HERTZ et al., 1993.
Pesquisa em Vizinhança Variável – (*Variable Neighborhood Search*) - VNS	Baseia-se em mudanças sistemáticas na estrutura de vizinhança, gera-se vizinhos com uma determinada estrutura de vizinhança e realiza-se uma busca local caso resulte em uma solução melhor do que a corrente, o processo é repetido com a mesma estrutura de vizinhança, caso contrário, um novo vizinho é gerado a partir de uma nova estrutura de vizinhança	MLADENOVIC; HANSEN, 1997; HANSEN, MLADENOVIC, 2001; HANSEN, MLADENOVIC, 2003.
***Greedy Randomized adaptive Search Procedure* – GRASP**	Facilita a geração de uma solução inicial boa, de melhor qualidade, utilizando a busca local apenas na tarefa de melhorar a solução inicial, baseando-se na utilização de diferentes soluções iniciais como ponto de partida para a busca local.	FESTAL; RESENDE, 2002.
***Simulated Anneling* - SA**	Baseia-se no princípio da física de recozimento de metais (*annealing*), ou seja, processo térmico que começa no estado liquido de um metal a temperaturas altíssimas, procedido pela gradativa diminuição de sua temperatura, até que se atinja o ponto de solidificação.	KIRKPATRICK; GELATT; VECCHI, 1983; METROPOLIS et al., 1953; BERTSIMAS; TSITSIKLIS, 1993; INGBER, 1993, HENDERSON; JACOBSON; JOHNSON, 2003; KOULAMAS; ANTONY; JAEN, 1994; BO; WODECKI, 2009; SOUZA, 2011.

Fonte: Autor

Quadro 2 – Descrição dos algoritmos heurísticos e metaheurísticos (cont.)

Algoritmo	Descrição	Referências
Colônia de Formigas (*Ant Colony Optimization*) - ACO	Utiliza a combinação do conhecimento cognitivo e coletivo, indivíduos pouco inteligentes, mas com comportamentos coletivos inteligentes, cada formiga utiliza uma comunicação indireta para indicar para as outras o quão bom foi o caminho que ela escolheu, para isso, elas espalham uma substância chamada “feromônio”, evidenciando o melhor caminho.	ZUBEN; ATTUX, 2008; WHITE; PAGUREK, 1998; DORIGO; MANIEZZO; COLORNI, 1996; FORSYSTH, 1997; HUANG; LIAO, 2004; JACKSON; BICAK; HOLCOMBE, 2008; PRABHAKAR; DEKTAR; GORDON, 2012.
Otimização Nuvem de Partículas (*Particle Swarm Optimisation*) - PSO	Baseia-se na inteligência de enxames, onde cada partícula ajusta seu voo de acordo com sua própria experiência e na de seus companheiros. A partir disso, faz uso de um grupo (população) de partículas que são inseridas em um espaço de solução para procurar um ótimo local. As partículas se comunicam entre si informando os valores da função objetivo em suas respectivas posições locais.	ZUBEN; ATTUX, 2008; KENNEDY; EBERHART, 1995a; KENNEDY; EBERHART, 1995 b; WHITE; PAGUREK, 1998; POLI; KENNEDY; BLACKWELL, 2007; IVERS, 2006; POLI, 2008; SHI; EBERHART, 1998; KULKARNI; VENAYAGAMOORTHY, 2011; MUSSI; NASHED; CAGNONI, 2011.
Otimização por Colônia de Abelhas Artificial (Artificial Bee Colony) - ABC	Baseia-se na busca inteligente das colônias de abelhas, simulando o comportamento das abelhas operárias. Trabalha com os conceitos de auto-organização e divisão do trabalho em colmeias.	KARABOGA, 2005; LI; PAN; TASGETIREN, 2014; TAHERI et al., 2011; AATHI; R, 2014; THAMMANO; PHU-ANG, 2013; ZHANG; WU, 2011; CUI; GU, 2015.

Fonte: Autor

Uma visão geral da aplicação do AG em diversas áreas, citando algumas referências, é apresentada no Quadro 3, onde percebe-se o vasto campo de utilização.

Quadro 3 – Aplicações do AG em diversas áreas

Utilização dos Algoritmos Genéticos	
Industria Mecânica	TAPLAK; ERKAYA; UZMAY, 2012; JIN; JEONG, 2014; PILTAN; SHIRI; GHADERI, 2012; CANYURT; OZTURK, 2008
Engenharia Elétrica	GHOLAMI; SHAHABI; HAGHIFAM, 2015; MORADI; ABEDINI, 2011; ALKHATIB; DUVEAU, 2012; YAS, 2011
Engenharia Civil	QUINIOU; MANDEL; MONIER, 2014; DONG et al., 2012; FAGHIHI; REINSCHMIDT; KANG, 2014; POURZEYNALI; SALIMI; KALESAR, 2013
Processamento de Imagens	LIU et al., 2015; SUMER; TURKER, 2013; HOSEINI; SHAYESTEH, 2012; AKGÜN; ERDO, 2015
Mineração de Dados	SHARMA, 2015; DONIS-DÍAZ et al., 2013; SANTHANAM; PADMAVATHI, 2015
Aprendizado de máquina	ALEXANDRE et al., 2014; YANG; XU; JIA, 2013; YANG et al., 2012
Medicina	YAN et al., 2013; MANTZARIS; ANASTASSOPOULOS; ADAMOPOULOS, 2011; PARK; CHUN; KIM, 2011; JABBAR; DEEKSHATULU; CHANDRA, 2013
Engenharia de Produção	MODOLO et al., 2015; GRASSI ; TRIGUIS; PEREIRA, 2014; WANG; LIU, 2013; GONÇALVES; RESENDE, 2015

Fonte: Autor

As técnicas utilizadas bem como a descrição das particularidades deste algoritmo metaheurístico, são abordadas no próximo capítulo.

3 ALGORITMOS GENÉTICOS

Este capítulo apresenta e discute os principais conceitos para o entendimento do funcionamento do AG. Inicia-se apresentando os conceitos básicos da genética, e a devida equivalência entre a biologia e o AG. Na sequência discorre sobre os operadores de cruzamento, seleção e mutação. Apresenta também as formas mais utilizadas para representar a solução do JSSP.

Os Algoritmos Genéticos representam um conjunto de técnicas que consistem na busca de soluções com base em mecanismos que imitam o processo da seleção natural, apresentada por Darwin. Em geral, diferem de métodos heurísticos por apresentarem características distintas: tratam e operam um conjunto de pontos conhecidos como população, ao invés de pontos isolados; operam num espaço de soluções codificadas e não diretamente no espaço de busca e, utilizam regras de transição probabilística, em vez de determinística (GOLDBERG, 1989).

A evolução da população para novas gerações (iterações) é realizada inserindo novos pontos (indivíduos ou cromossomos) na população atual usando operadores genéticos: cruzamento (*crossover*) e mutação. O operador de cruzamento determina o mecanismo que combina dois ou mais cromossomos existentes para criar dois ou mais filhos; o operador de mutação promove mudanças aleatórias nos cromossomos, a fim de abranger o máximo das opções e garantir o acesso probabilístico a todo espaço de busca. Esse procedimento é definido pela analogia com conceitos básicos da genética, conforme abordado na próxima seção.

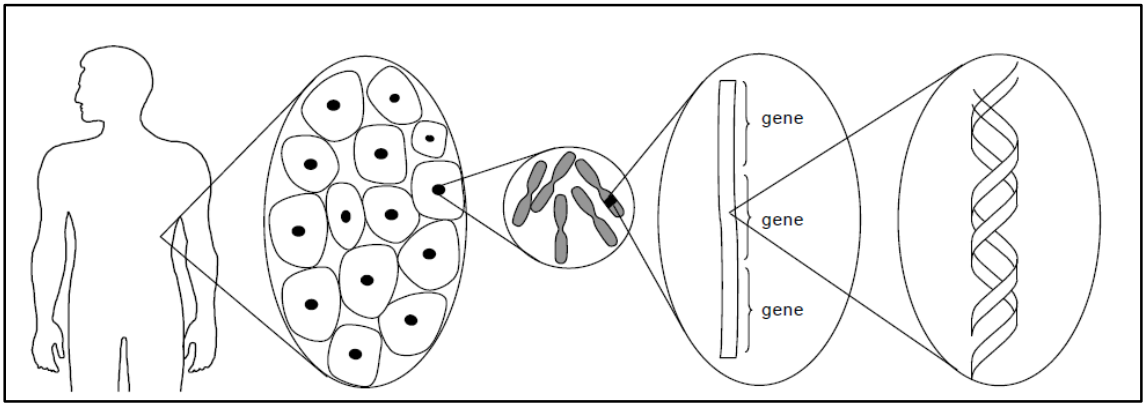
3.1 CONCEITOS BÁSICOS DA GENÉTICA

Percebe-se que os indivíduos dentro de um ecossistema competem entre si por recursos limitados, tais como comida, água, território, acasalamento, entre outros (VAN WYHE, 2009).

Ocorrendo a competição, indivíduos (animais, vegetais) de uma mesma espécie menos aptos, não terão êxito, pois terão menos chances de reprodução ou até a extinção no caso de um grupo todo não conseguir os suprimentos e entrar em desequilíbrio (DARWIN, 1953). Esta descendência reduzida faz com que a probabilidade de serem propagados os genes do indivíduo ao longo de sucessivas gerações seja menor (KLUG; CUMMINGS; SPENCER, 2006).

Os genes, unidade básica da informação genética, são responsáveis pela transmissão de características do novo indivíduo. A combinação entre os genes, com características positivas, dos indivíduos que sobrevivem produzirá um novo indivíduo que poderá ser melhor adaptado às características de seu meio ambiente. (KLUG; CUMMINGS; SPENCER, 2006).

Figura 10 – Ampliação de um organismo focando o material genético



Fonte: Castro (2005)

Como visualizado na Figura 10, percebe-se uma ampliação do indivíduo para enfocar o material genético: organismo formado por células; células contendo os cromossomos; cromossomos constituídos por genes; genes formados de cadeias de DNA. O DNA (***d****eoxyribo****n****ucleic* ***a****cid* ou ácido

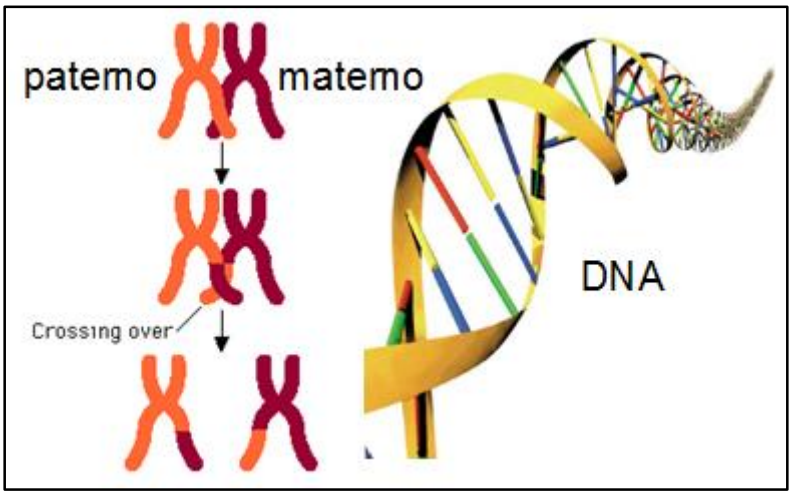
desoxirribonucleico, em português) é uma molécula de proteína que codifica toda a informação necessária para existência e composição dos organismos (WALKER, 2002).

O conjunto completo de material genético (todos os cromossomos) é chamado de genoma, por sua vez um conjunto específico de genes no genoma é chamado de genótipo. O genótipo é base do fenótipo, que é a expressão das características físicas e mentais codificadas pelos genes e modificadas pelo ambiente, tais como cor dos olhos, inteligência, altura, etc. (WALKER, 2002).

3.1.1 Mecanismos de reprodução

Em uma reprodução a troca do material genético entre os indivíduos transcorre através do cruzamento dos cromossomos pertencentes a cada pai. Cada pai num processo de cruzamento forma uma nova combinação cromossômica, conforme visualizado na Figura 11.

Figura 11 – Cruzamento cromossômico e filamento de DNA

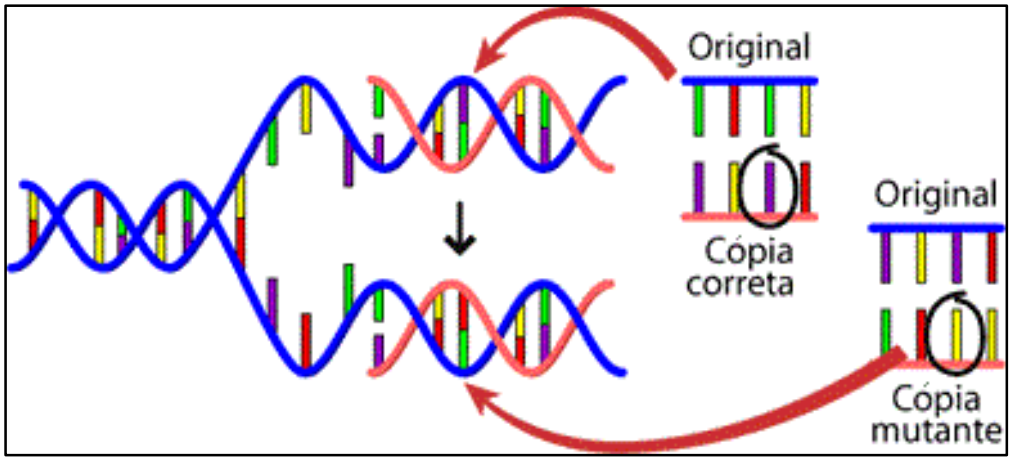


Fonte: www.estudopratico.com.br (2015)

No processo de reprodução e troca do material genético, a replicação do DNA é extremamente complexa. Consequentemente, podem ocorrer pequenos

erros ao longo do tempo, gerando mutações dentro do código genético como visualizado na Figura 12. Estas mutações podem ser boas, ruins ou neutras (KLUG; CUMMINGS; SPENCER, 2006).

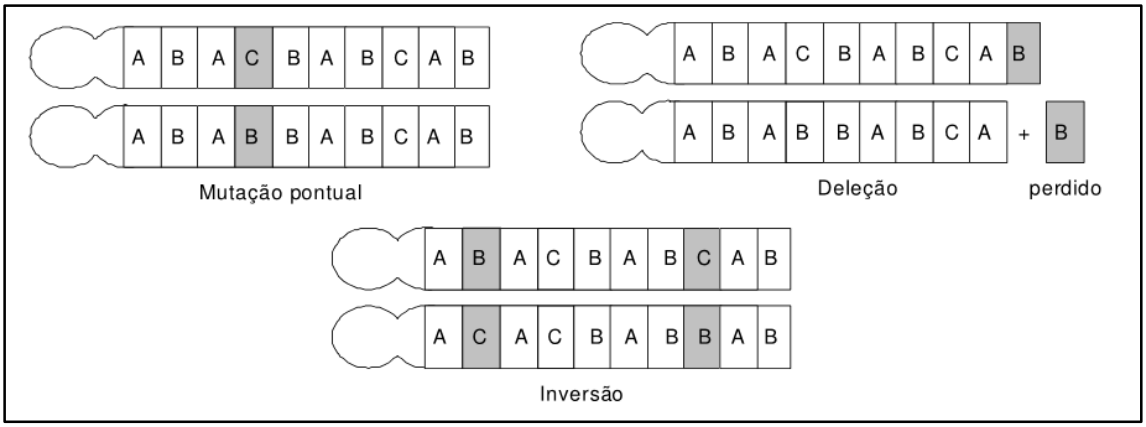
Figura 12 – Mutação no cromossomo



Fonte: www.evolution.berkeley.edu (2015)

As mutações do material genético na cadeia do DNA podem ocorrer de muitas formas. As mais conhecidas são: pontualmente na cadeia, perda de alguma informação da cadeia e inversão de posições dentro da cadeia (VON ZUBEN, 2000), estes tipos de mutações são visualizados na Figura 13.

Figura 13 – Ilustração de diferentes tipos de mutação



Fonte:Von Zuben, (2000)

3.2 ALGORITMOS EVOLUTIVOS

Algoritmos evolutivos (AE) se baseiam nos conceitos evolutivos (VON ZUBEN, 2000; SIVANANDAM; DEEPA, 2007), aplicam técnicas heurísticas e metaheurísticas que imitam a reprodução, impõem variações aleatórias, promovem competição e executam a seleção de indivíduos de uma determinada população (MATTFELD, 1996).

Para facilitar a compreensão, no Quadro 4 visualiza-se a equivalência entre a nomenclatura utilizada pela genética e a utilizada por AE.

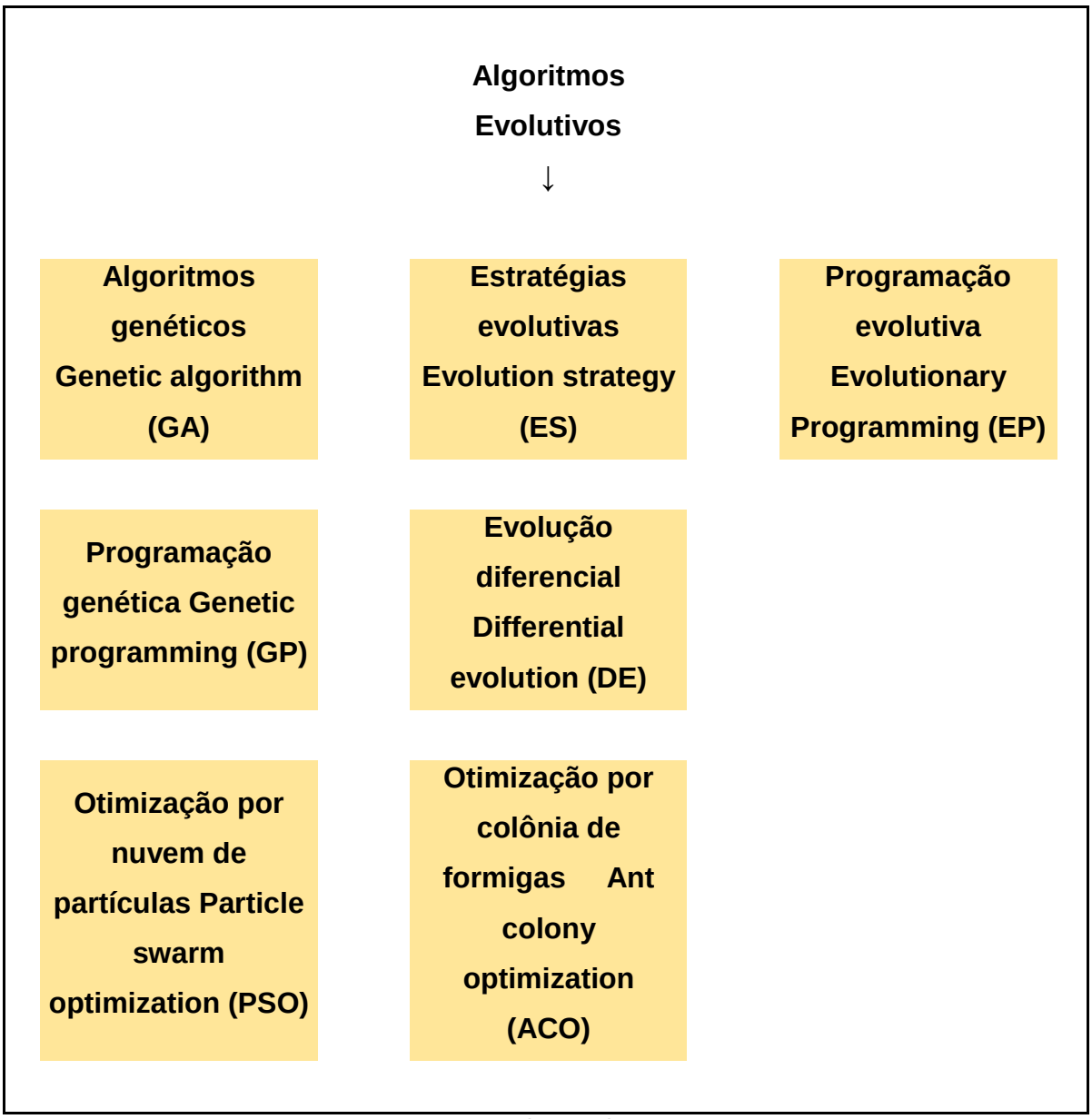
Quadro 4 – Equivalência Da Biologia e Algoritmos Evolutivos

Biologia (Genética)	**Algoritmos Evolutivos**
Cromossomo	Estrutura de dados
Gene	Um ou mais atributos ocupando uma determinada posição da estrutura de dados
Lócus	Posição ocupada por um gene em uma estrutura de dados
Alelo	Diferentes valores (dados) que um gene pode assumir
Crossover	Troca de porções entre duas ou mais estruturas de dados
Mutação	Variação aleatória em uma estrutura de dados individual
Fitness	Valor que quantifica a qualidade relativa ou absoluta (nível de adaptação) de um indivíduo; retorno da função de avaliação
Seleção	Processo que permite a sobrevivência e reprodução dos indivíduos mais aptos em detrimento dos menos aptos
Genótipo	Estrutura de dados que representa uma solução candidata de um determinado problema
Fenótipo	Decodificação de um ou mais cromossomos que leva a um ponto do espaço de busca

Fonte: Autor

Existe vasta literatura sobre a utilização e classificação dos AE, (CÉSAR TREJO ZÚÑIGA; LÓPEZ CRUZ; GARCÍA, 2014; WANG; SANIN; SZCZERBICKI, 2015), os conceitos dos AE incorporam os fundamentos de outros algoritmos tais como: Algoritmos Genéticos, Estratégias evolutivas, Programação genética entre outros (GAO; SHAO, 2014; PASCUAL et al.; GONG et al., 2015; SIVANANDAM; DEEPA, 2007) conforme é visualizado na Figura 14.

Figura 14 – Distribuição dos Algoritmos Evolutivos



Fonte: Gong, 2015; Sivanandam; Deepa, 2007

3.3 PRINCÍPIOS BÁSICOS DO ALGORITMO GENÉTICO

O AG utiliza os conceitos dos algoritmos evolutivos, que funcionam mantendo uma população de estruturas, chamadas de indivíduos ou cromossomos, operando semelhantemente a seleção natural. Nestas estruturas,

aplicam-se os operadores genéticos de cruzamento e de mutação, que são os principais mecanismos de busca para explorar regiões desconhecidas (MITCHELL, 1995; MITCHELL,1998). Cada indivíduo recebe uma avaliação que quantifica sua qualidade em relação ao todo de soluções. Repetindo-se o processo, com base nesta avaliação de forma a simular a sobrevivência do mais apto.

Especificamente, os operadores genéticos são aproximações computacionais de fenômenos que ocorrem na genética, tais como a reprodução sexuada, a mutação, etc. Eventualmente, após uma série de gerações que passaram pelo processo de seleção, os melhores indivíduos prevalecerão (LINDEN, 2012; SIVANANDAM; DEEPA, 2007).

O esquema básico utilizado pelo AG e apresentado por Linden (2012), é definido da seguinte forma:

a) Inicialize a população de cromossomos.
b) Avalie cada cromossomo na população.
c) Selecione os pais para gerar novos cromossomos.
d) Aplique os operadores de cruzamento e mutação a estes pais de forma a gerar os indivíduos da nova geração.
e) Apague os velhos membros da população.
f) Avalie todos os novos cromossomos e insira-os na população.
g) Se o tempo acabou, ou o melhor cromossomo satisfez os requerimentos e desempenho, retorne-o, caso contrário, volte para o passo c).

3.3.1 Codificação e representações no AG

Existe uma relação direta e também uma sutil diferença entre representação e codificação. Muitas vezes esses termos são usados como sinônimos não ficando bem claro na literatura essa diferenciação. Uma mesma codificação pode dar suporte ou origem a diversas representações, sendo o

contrário também verdadeiro, pode-se ter a representação de uma solução codificada de diferentes maneiras.

Uma das etapas mais críticas no AG são as formas como são representadas as soluções e seus respectivos operadores genéticos. Essas representações são dependentes do problema e, no caso específico do JSSP uma definição inadequada pode ocasionar um aumento no número de soluções não factíveis, especialmente quando indivíduos são modificados pelos operadores genéticos ou operadores de cruzamento ou operadores de mutação (VON ZUBEN, 2000).

Basicamente a codificação está relacionada com o sistema de numeração utilizado (bases decimais, binária, hexadecimal, etc.), em problemas que exigem respostas inteiras (discretas, como sequências) ou reais (continuas). O processo de codificação pode ser realizado usando bits, números, árvores, matrizes, listas ou quaisquer outros objetos, na sequência é apresentado as três codificações mais utilizadas.

3.3.1.1 Codificação Binária

A codificação binária é a abordagem mais tradicional e muito utilizada pelos pesquisadores, não só por ter sido a primeira a aparecer no modelo apresentado por Holland (1975), mas também pela sua simplicidade e rastreabilidade (MAN; TANG; KWONG, 1996). Em muitos casos, a codificação binária é utilizada mesmo quando as variáveis do problema são inteiras ou reais (pode ser codificada utilizando-se *strings* binárias). A principal motivação para o uso desta codificação vem da teoria dos esquemas (*schemata theory*) (HOLLAND, 1975; 1992). Sendo fácil de usar e manipular, teoricamente simples de analisar e não há uniformidade nos operadores.

Uma característica peculiar da codificação binária é o fato de que dois pontos vizinhos na representação necessariamente não são vizinhos no espaço de busca definido pelo problema. Para amenizar esta característica, muitas vezes, faz-se uso do código Gray garantindo a mudança entre a vizinhança que

será apenas de 1 *bit*. Hollstien (1971) investigou o uso de AG para funções de duas variáveis e afirmou: uma representação utilizando o código Gray tem desempenho melhor que a codificação binária normal.

3.3.1.2 Codificação Real

Codificação real dos indivíduos é feita através de *string* de números reais. Ela é normalmente utilizada para otimização de parâmetros contínuos, situação na qual a codificação binária não é muito adequada, pois seriam necessários muitos *bits* para obter boa precisão numérica.

3.3.1.3 Codificação Inteira

Na codificação inteira a representação dos indivíduos é feita através de *string* de estados, devendo ser distintos dos utilizados com representação binária ou real e podendo permutar entre si para achar a melhor solução.

Na analogia com a genética, codificação define as soluções no espaço do genótipo, constitui os genes que codificam características das soluções do problema, que são expressas pela representação adotada (fenótipo). Representação e codificação são conceitos dependentes, mas não necessariamente iguais.

Na Figura 15 são visualizados os cromossomos nas codificações binária, octal, hexadecimal, inteira e real.

Figura 15 – Exemplos de codificações

Codificação	Representação do cromossomo									
Binária	Pai 1	1	0	0	0	1	1	0	1	0	1
	Pai 2	1	1	1	1	1	0	0	1	0	0
Octal	Pai 1	7	4	6	5	0	1	2	3	0	7
	Pai 2	6	5	4	3	6	0	2	5	1	2
Hexadecimal	Pai 1	A	B	0	9	C	4	9	7	F	E
	Pai 2	F	F	1	2	3	D	0	8	7	5
Inteira	Pai 1	7	8	9	0	1	2	3	4	6	5
	Pai 2	9	7	0	1	4	5	6	3	2	8
Real	Pai 1	0.12	1.14	2.17	5.18	0.99	3.14	3.99	1.50	1,99	4.87
	Pai 2	6.27	0.31	0.77	4.33	3.51	6.00	2.89	4.17	0.01	1.05

Fonte: Autor

Diferentes representações para o JSSP são desenvolvidas usando a mesma codificação no AG, como pode ser visto na seção a seguir.

3.4 FORMAS DE REPRESENTAÇÃO DA SOLUÇÃO DO JSSP NO AG

Devido à flexibilidade do AG e sua vasta utilização em diversos problemas de sequenciamento, várias formas de representações foram desenvolvidas (ABDELMAGUID, 2010; CHENG, 1996). Essas representações podem ser classificadas em duas grandes abordagens: as representações diretas e as indiretas.

Quando o problema é de otimização contínua, trabalhando com números reais, adota-se uma codificação real para representar a solução, neste caso tem-se uma **representação direta**; ou ainda pode utilizar uma codificação binária para representar a solução, mas neste caso tem-se uma **representação**

indireta, pois para se obter a solução final é preciso uma etapa de conversão, binária para decimal, por exemplo.

Diferentes e usuais formas de representação da solução do JSSP apresentadas por Cheng (1996) e Abdelmaguid (2010), são a seguir discutidas, separadas segundo a codificação adotada no AG.

3.4.1 Representações baseadas na codificação inteira

a) Representação baseada em operações (OB)

A representação baseada em operações, foi proposta por Gen et al. (1994) e consiste numa *string* de tamanho n x m, onde n representa a quantidade de *job*s e m a quantidade de máquinas. Nesta representação cada *job* aparece exatamente m vezes, representando assim as devidas operações que serão efetuadas para completar totalmente o *job*. Observa-se que existe uma sequência de produção pré-definida dependendo da sequência tecnológica, ou melhor, a sequência exigida para se obter determinado produto (KUBOTA, 1995).

O fato é que neste tipo de representação ocorrerá uma permutação de *n job*s, com m repetições, gerando um vetor no qual é aplicado os operadores de cruzamento. Como a interpretação do cromossomo depende do contexto da operação, qualquer permutação no cromossomo gera uma sequência factível.

Na Figura 16, é visualizado a sequência tecnológica de um *job shop* formado por 3 *job*s e 3 máquinas. Adotando-se o cromossomo C= {1,2,2,3,3,1,1,3,2} para representar a ordem e as sequências das operações, define-se cada posição como sendo a operação $O_{jim,}$ na qual j representa o *job*, i indica a sequência da operação e m, a máquina. O cromossomo C pode ser traduzido como uma lista de ordens de operações e representa a Solução S, que ficará como se segue:

S= {$O_{111;}$ O_{213}; O_{221}; O_{312}; O_{323}; O_{123}; O_{132}; O_{331}; O_{232}}, para facilitar a interpretação à operação O_{323} está na cor verde e a operação O_{232} está da cor azul e ambas aparecem na Figura 16 marcadas com suas respectivas cores para maior entendimento.

Figura 16 – Sequência tecnológica *job shop* 3X3

	Máquina (Tempo de processamento)		
Job1	M1(1)	M3(2)	M2(1)
Job2	M3(1)	M1(2)	M2(1)
Job3	M2(1)	M3(3)	M1(1)

Fonte: Autor

Como se pode facilmente observar este tipo de representação usa a codificação inteira.

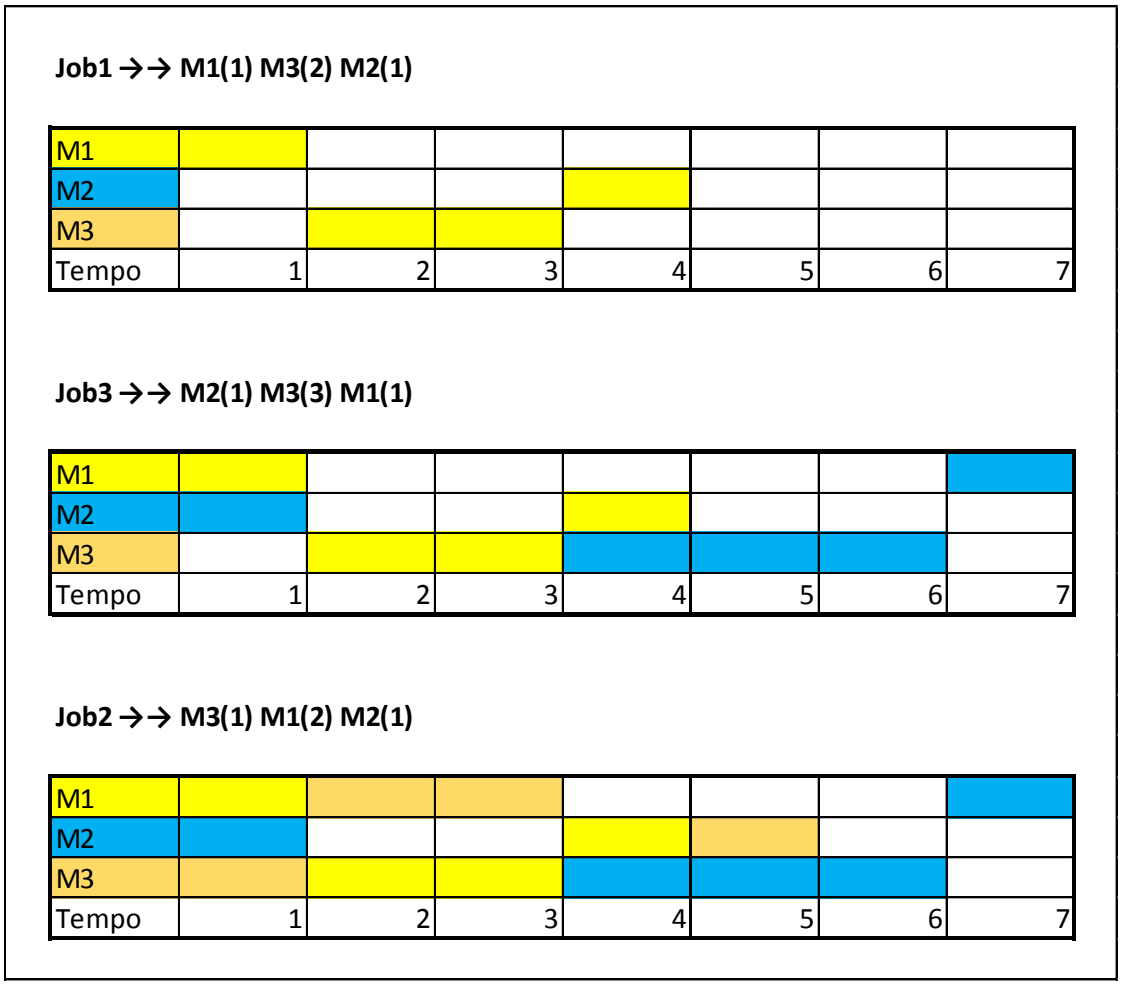
b) Representação baseada em *job* (JB)

A representação baseada em *job*s, foi utilizada por Holsapple et al (1993), para lidar com os problemas de sequenciamento estático no contexto de produção flexível. Consiste numa *string* de tamanho n, onde n representa a quantidade de *job*s. Usando a ordem dos *job*s apresentados no cromossomo, o AG é utilizado para marcar todas as operações do *job*, dando sequência em todas as máquinas.

Para sequenciar uma operação, o AG procura por um espaço de tempo vazio na máquina, atribuído o mesmo sem violar as restrições tecnológicas. Uma representação da solução, seguindo a sequência tecnológica da Figura 16, poderia ser S= {1,3,2}, o AG preencheria toda a sequência das operações do *job* 1, nos espaços de tempo, depois nos espaços de tempo vazios do *job* 3 e finalmente nos espaços tempo que sobraram do *job* 2.

Nota-se que na Figura 17 é visualizado o tempo de execução de cada operação em parênteses, além da sequência tecnológica.

Figura 17 – Exemplo das etapas da representação JB



Fonte: Adaptado de Abdelmaguid, 2010

Para o *job* 1 tem-se a sequência (M1, M3, M2), com seus respectivos tempos 1 unidade de tempo (u.t) para M1, 2 u.t para M3 e 1 u.t para M2, e assim sucessivamente para cada *job*. Na Figura 17 visualiza-se como seria uma possível sequência de preenchimento.

Assim como na representação OB, essa representação também usa a codificação inteira.

c) Representação baseada em lista de preferências (LISTA)

A representação baseada em lista de preferências foi originalmente apresentada por Davis (1985), para alguns tipos de problemas de sequenciamento. Consiste num cromossomo de tamanho n x m, dividido em m subcromossomos, um para cada máquina, estes subcromossomos representam as listas de preferência que deverão ser obedecidas em cada máquina, por exemplo, para um *job shop* 3 x 3, dado o seguinte cromossomo, C= {(3,2,1), (1,2,3), (2,3,1)}, a leitura do mesmo segue-se assim:

- Para a máquina 1, a lista de preferências é (3,2,1) onde o *job* 3 tem preferência;
- Para a máquina 2, a lista de preferências é (1,2,3) onde o *job* 1 tem preferência;
- Para a máquina 3, a lista de preferências é (2,3,1) onde o *job* 2 tem preferência.

d) Representação baseada em tempo de conclusão

A representação baseada no tempo de conclusão foi proposta por Yamada e Nakano (1997). Para representar o cromossomo utiliza-se uma cadeia de números inteiros que representam uma lista ordenada do tempo de conclusão de cada operação, o valor inteiro armazenado em cada gene representa a ordem do tempo de conclusão de cada operação. Esta representação por necessitar de um esforço computacional muito grande, para corrigir uma quantidade enorme de sequências não factíveis não foi comparada no trabalho apresentado por Abdelmaguid (2010).

Para o exemplo dado na Figura 18, tem-se 3 *job*s e 3 máquinas, um cromossomo pode ser representado por C_{jir}, onde alelo mostra o tempo de conclusão para cada operação, o índice j representa o *job*, o índice i a operação e o índice r a máquina.

Figura 18 – Exemplo de sequência tecnológica para *job shop* 3X3

	Máquina (Tempo de processamento)		
Job1	M1(1)	M3(2)	M2(2)
Job2	M3(2)	M1(2)	M2(1)
Job3	M2(1)	M3(3)	M1(1)

Fonte: Autor

C = [C_{111}, C_{123}, C_{132}, C_{213}, C_{221}, C_{232}, C_{312}, C_{323}, C_{331}];

C = [1, 2, 2, 2, 2, 1, 1, 3, 1];

e) Representação baseada por máquina - *machine-based* (MB)

A representação baseada por máquina foi proposta por Dorndorf e Pesch (1995), onde o cromossomo, do tamanho do número de máquinas, representa uma sequência da ordem de máquinas que serão utilizadas para determinado *job*. Para se tratar esse tipo de representação foi apresentada a heurística *shifting bottleneck*, por Adams et al. (1988) A heurística *shifting bottleneck* sequência as máquinas uma a uma sucessivamente, identificando qual é o gargalo entre as máquinas que ainda não foram sequenciadas, recalculando e otimizando as sequências a cada rodada.

A relação entre a heurística *shifting bottleneck* e o AG, é que o gargalo não é um critério de decisão para a escolha da próxima máquina, esta escolha é controlada pelos cromossomos gerados pelo AG o qual busca a melhor sequência, e somente depois são tratados pelo *shifting bottleneck*.

No trabalho de Ivers (2006), o autor observa que este tipo de representação gera muitas sequências não factíveis, gastando muito tempo para converter sequências não factíveis em factíveis. O trabalho de Abdelmaguid (2010) mostra que representações por ordem de despacho e por máquinas demoram muito mais tempo utilizando o AG, em relação a outras representações.

f) Representação baseado em regra de prioridade

A representação baseada em regra de prioridade foi apresentada por Dorndorf e Pesch (1995). Segundo Cheng (1996), este tipo de representação na prática tem sido frequentemente a mais utilizada para resolver problemas de sequenciamento, devido a sua facilidade de implementação e menor complexidade. O cromossomo consiste em uma série de regras de despacho prioritárias, que serão executadas em sequência conforme a prioridade, para assim gerar as ordens das operações. O tamanho do cromossomo é igual ao número total de operações de um determinado problema, n x m. A decodificação do cromossomo em solução utiliza o algoritmo desenvolvido por Giffler e Thompson (1960).

3.2.2 Representações baseadas na codificação binária

a) Representação baseada na relação de pares de *job*s

A representação baseada na relação de pares de *job*s, apresentada por Nakano e Yamada (1991), utiliza uma matriz binária para codificar o sequenciamento, o qual é determinado de acordo com comparação de precedência dos pares de *job*s em máquinas correspondentes.

A representação, segundo a literatura, é uma das mais complexas entre as representações para JSSP e altamente redundante segundo Fang e Ross (1993), produzindo um número grande de soluções não factíveis, após aplicação dos operadores de cruzamento, e soluções redundantes devido a permutação dos *job*s. Cada alelo representa a prioridade do *job* em relação ao seu par, cada posição da matriz é representada pela variável X_{ijm}, onde o índice i representa o *job* i, o índice j representa o *job* j e índice *m* representa a máquina nas quais ambos serão comparados, considerando sempre a sequência operacional da

máquina como prioridade na comparação. Sendo assim, cada elemento da matriz será preenchido da seguinte maneira: assumira o valor 1 se o *job* i tiver precedência ao *job* j na máquina m, e 0 casos diferente.

Para um problema de 3 *job*s e 3 máquinas, seguindo a sequência operacional e a prioridade operacional apresentadas na Figura 19 e efetuando as comparações par a par, como se segue:

Figura 19 – Exemplos de sequência tecnológica e prioridade operacional para *job shop* 3X3

	Sequência operacional		
job1	m1	m3	m2
job2	m2	m1	m3
job3	m3	m1	m2

	Prioridade operacional		
m1	job1	job2	job3
m2	job2	job3	job1
m3	job2	job1	job3

Fonte: Autor

(j_1, j_2) nas máquinas (m_1, m_3, m_2) → X_{121}, X_{123}, X_{122};

(j_1, j_3) nas máquinas (m_1, m_3, m_2) → X_{131}, X_{133}, X_{132};

(j_2, j_3) nas máquinas (m_2, m_1, m_3) → X_{232}, X_{231}, X_{233};

Obtém-se a matriz (3):

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad (3)$$

Como apresentado poder-se-iam ter outras comparações permutando os pares de *job*s, como o exemplo seguinte mostra.

(j_2, j_3) nas máquinas (m_2, m_1, m_3) → X_{232}, X_{231}, X_{233};

(j_2, j_1) nas máquinas (m_2, m_1, m_3) → X_{212}, X_{211}, X_{213};

(j_1, j_3) nas máquinas (m_1, m_2, m_3) → X_{131}, X_{132}, X_{133};

Deste novo exemplo obtém-se a matriz (4):

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad (4)$$

b) Representação baseada em gráfico disjuntivo

A representação baseada em gráfico disjuntivo, apresentada por Tamaki e Nishikawa (1992), é formada por um vetor binário de tamanho **(m** x **n** x **(n- 1) /2)**. Pode ser vista como uma representação baseada em relação de pares de *job*s. Este tipo de representação é visual e apresenta facilmente a sequência tecnológica bem como as restrições do problema. Baseia-se na teoria dos grafos, sendo definida pelo o grafo G = (**N, A, E**), onde **N** representa os nós que indicam cada operação, **A**, representada pelas setas continuas, indicam a sequência tecnológica de cada *job*, e **E,** representada pelas setas tracejadas, indicam a ordem de possibilidades de prioridades das operações que serão processados pela mesma máquina.

Figura 20 – Exemplo de sequência tecnológica para *job shop* 3X3

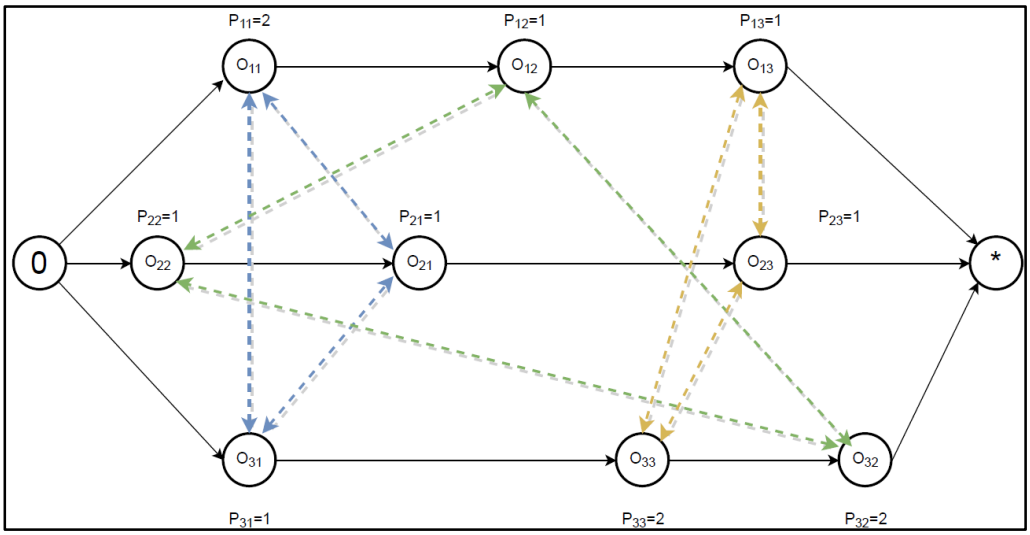
	Sequência Tecnologica		
Job1	M1(2)	M2(1)	M3(1)
Job2	M2(1)	M1(1)	M3(1)
Job3	M1(1)	M3(2)	M2(2)

Fonte: Autor

A orientação das setas tracejadas dos arcos disjuntos são os alelos do cromossomo C, formadas pelas sequências de operações distintas na mesma máquina, obedecendo a seguinte regra de preenchimento: valor 0 quando a orientação for de $\mathbf{O_{jm}}$ para $\mathbf{O_{km}}$, onde **j** representa o *job* antecessor do *job* **k (j < k),** e **m** a máquina, e valor 1 quando o inverso. Para formar o cromossomo considera-se a sequência crescente das máquinas.

Para exemplificar, na Figura 20 é possível visualizar uma determinada sequência tecnológica do problema 3 x 3, representada pelo diagrama disjunto da Figura 21.

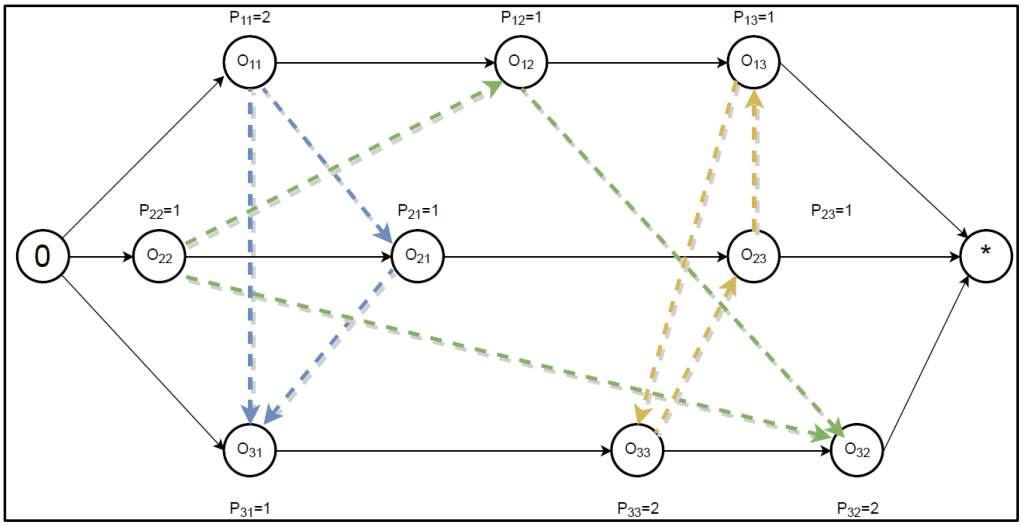
Figura 21 – Diagrama disjunto para o problema 3x3



Fonte: Autor

Supondo o cromossomo C = {0, 0, 0, 1, 0, 0, 1, 0, 1}, o diagrama orientado resultante é visualizado na Figura 22.

Figura 22 – Diagrama orientado para o problema 3x3



Fonte: Autor

c) Representação por semente dinâmica

Recentemente, Grassi (2014) propôs um Algoritmo Genético com codificação binária baseada no uso de um conceito chamado de semente dinâmica (SD) para produzir as soluções. A abordagem faz uso dos operadores canônicos, originalmente desenvolvidos para a representação binária, visando aproveitar as vantagens desses operadores.

Para se ter uma noção do funcionamento do SD, na Figura 23 tem-se o exemplo de rotas para o problema teórico *job shop* LA01 com 10 jobs e 5 máquinas (LAWRENCE,1984) onde J_n representa o *job* n e M_n representa cada máquina que será visitada pelo *job n* na ordem apresentada em cada linha. Exemplificando, o job J1 para a sua execução completa deverá passar pela máquina M2 em seguida pela M1, M5, M4 e por último na M3, após esta sequência o produto 1 estará concluído. Este raciocínio é idêntico para os outros *jobs.*

Figura 23 – Rotas dos *jobs* para o problema LA01

J1	**M2**	**M1**	**M5**	**M4**	**M3**
J2	**M1**	**M4**	**M5**	**M3**	**M2**
J3	**M4**	**M5**	**M2**	**M3**	**M1**
J4	**M2**	**M1**	**M5**	**M3**	**M4**
J5	**M1**	**M4**	**M3**	**M2**	**M5**
J6	**M2**	**M3**	**M5**	**M1**	**M4**
J7	**M4**	**M5**	**M2**	**M3**	**M1**
J8	**M3**	**M1**	**M2**	**M4**	**M5**
J9	**M4**	**M2**	**M5**	**M1**	**M3**
J10	**M5**	**M4**	**M3**	**M2**	**M1**

Fonte: Adaptado Grassi, 2014

O primeiro passo dessa representação é converter a matriz da Figura 23 em outra, que represente as máquinas, sempre a partir da máquina um (M1) e assim em diante e em cada linha segundo a ordem dos *job*s, conforme aparecem em cada máquina, e também seguindo a ordem da Figura 23, de cima para baixo e da esquerda para a direita, conforme visto na matriz da Figura 24.

Figura 24 – Semente – Sequência de *jobs* por máquina para o problema LA01

M1	**J2**	**J5**	**J1**	**J4**	**J8**	**J6**	**J9**	**J3**	**J7**	**J10**
M2	**J1**	**J4**	**J6**	**J9**	**J3**	**J7**	**J8**	**J5**	**J10**	**J2**
M3	**J8**	**J6**	**J5**	**J10**	**J2**	**J3**	**J4**	**J7**	**J1**	**J9**
M4	**J3**	**J7**	**J9**	**J2**	**J5**	**J10**	**J1**	**J8**	**J4**	**J6**
M5	**J10**	**J3**	**J7**	**J1**	**J2**	**J4**	**J6**	**J9**	**J5**	**J8**

Fonte: Adaptado Grassi, 2014

Esta nova matriz baseada agora na sequência dos jobs em cada máquina é chamada de semente, sempre ordenada pela M1 em diante. O nome semente é escolhido, pois esta é a primeira solução usada pelo AG para iniciar o processo

de busca. As demais soluções geradas, a partir desta semente, serão soluções já otimizadas. Com isso se espera que o processo encontre soluções com *makespan* menores.

No passo seguinte é gerado pelo AG, o cromossomo de tamanho **n x (m-1)**, uma matriz binária com valores aleatórios, como visualizado na Figura 25. Este cromossomo binário será utilizado para operar as devidas permutações na semente da Figura 24, deste ponto em diante novas permutações ocorrerão, gerando outra solução.

Figura 25 – Cromossomo gerado pelo AG

0	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**
1	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
0	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**

Fonte: Autor

A permutação consiste no seguinte raciocínio: comparando cada linha da matriz das máquinas com a matriz do cromossomo, em cada linha da semente só ocorrerá a permutação entre os *job*s adjacentes daquela máquina se e somente se, o valor na célula do cromossomo for igual a um, caso contrário o *job* fica na mesma posição (ordem). Este processo é visualizado na Figura 26.

Figura 26 – Interpolação da semente e do cromossomo

M1	J2	0	J5	1	J1	1	J4	0	J8	0	J6	0	J9	1	J3	0	J7	1	J10
M2	J1	1	J4	1	J6	0	J9	0	J3	1	J7	0	J8	1	J5	0	J10	0	J2
M3	J8	0	J6	0	J5	0	J10	1	J2	1	J3	0	J4	0	J7	0	J1	1	J9
M4	J3	1	J7	1	J9	1	J2	1	J5	0	J10	0	J1	1	J8	0	J4	0	J6
M5	J10	0	J3	1	J7	1	J1	0	J2	0	J4	1	J6	1	J9	1	J5	0	J8

Fonte: Autor

Obtida a nova matriz permutada, baseada no cromossomo binário, visualizada na Figura 27, esta por sua vez é convertida numa matriz que representa as prioridades dos *job*s nas máquinas, ou seja, a solução, como visualizado na Figura 28.

Figura 27 – Matriz das permutações da semente baseada no cromossomo

M1	**J2**	**J1**	**J4**	**J5**	**J8**	**J6**	**J3**	**J9**	**J10**	**J7**
M2	**J4**	**J6**	**J1**	**J9**	**J7**	**J3**	**J5**	**J8**	**J10**	**J2**
M3	**J8**	**J6**	**J5**	**J2**	**J3**	**J10**	**J4**	**J7**	**J9**	**J1**
M4	**J7**	**J9**	**J2**	**J5**	**J3**	**J10**	**J8**	**J1**	**J4**	**J6**
M5	**J10**	**J7**	**J1**	**J3**	**J2**	**J6**	**J9**	**J5**	**J4**	**J8**

Fonte: Adaptado Grassi, 2014

Figura 28 – Transformação da Matriz em prioridades de *jobs* por máquinas

	J1	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**	**J9**	**J10**
M1	**2°**	**1°**	**7°**	**3°**	**4°**	**6°**	**10°**	**5°**	**8°**	**9°**
M2	**3°**	**10°**	**6°**	**1°**	**7°**	**2°**	**5°**	**8°**	**4°**	**9°**
M3	**10°**	**4°**	**5°**	**7°**	**3°**	**2°**	**8°**	**1°**	**9°**	**6°**
M4	**8°**	**3°**	**5°**	**9°**	**4°**	**10°**	**1°**	**7°**	**2°**	**6°**
M5	**3°**	**5°**	**4°**	**9°**	**8°**	**6°**	**2°**	**10°**	**7°**	**1°**

Fonte: Adaptado Grassi, 2014

A nova matriz, com as prioridades dos *job*s em cada máquina, é avaliada pela função de aptidão, (rotina em linguagem C, que simula o sequenciamento) para calcular o novo *makespan*. Caso os critérios de parada não tenham sido atingidos o processo se repetirá.

Após certa quantidade de avaliações, sem melhora, escolhe-se outra semente baseada na melhor solução encontrada. Nesta rodada, e com este novo valor, efetua-se uma nova rodada com uma semente inicial melhorada, daí o nome semente dinâmica. Segundo as pesquisas de Grassi, Triguis e Pereira

(2014), o processo de convergência se comporta melhor ou converge mais rapidamente encontrando valores melhores, quando temos uma semente “boa”.

.

3.4.3 Representações baseadas na codificação real

a) Representação por chaves aleatórias

Os algoritmos genéticos utilizando chaves aleatórias (*Randon Keys*) foram apresentados pela primeira vez por Bean (1994), para problemas de sequenciamento, onde os indivíduos são representados como uma cadeia de permutação. Neste tipo de representação, os cromossomos são apresentados como *strings* de números reais gerados aleatoriamente no intervalo entre 0 e 1, muito semelhante a representação baseada em operações (OB), com a diferença que cada gene é preenchido com números gerados aleatoriamente.

Figura 29 – Processo de atribuição do RK

Pai 1	0,25	0,19	0,67	0,05	0,89
crossover					
Pai 2	0,63	0,9	0,76	0,93	0,08
	S1	S2	S3	S4	S5
Filho	0,25	0,9	0,76	0,05	0,89
	S4	S1	S3	S5	S2
Ordem Crescente	0,05	0,25	0,76	0,89	0,9
Sequência da Operação	4	1	3	5	2

Fonte: Autor

Após o devido cruzamento entre os cromossomos, os números aleatórios da nova geração são classificados e a ordem resultante é utilizada para substituir esses números com um número inteiro (a ordem). Para cada operação é atribuído um valor inteiro, sendo assim a cadeia resultante de inteiros é

equivalente a uma série de operações. Essa cadeia é interpretada numa sequência viável. Na Figura 29 é possível visualizar como ocorre o processo de atribuição das chaves aleatórias.

Existe vasta literatura sobre a utilização de AG com *Randon Keys*, aplicados a problemas de sequenciamento, telecomunicações, corte de peças entre outros (GONÇALVES; RESENDE; RUIZ et al., 2015; LALLA-RUIZ et al., 2014; GONÇALVES; RESENDE, 2013).

Todas as representações apresentadas anteriormente são a seguir reunidas resumidamente no Quadro 5, separadas por tipo de codificação e com indicação de autores.

Quadro 5 – Diferentes representações agrupadas por tipo de codificação

Representação	Codificação	Autor (es)
Representação baseada em operação (OB)	Codificação Inteira	GEN et al. 1994; KUBOTA, 1995, FAN et al 1993.
Representação baseada em *job* (JB)		HOLSAPPLE et al, 1993.
Representação baseada em lista de preferências (LISTA)		DAVIS,1985.
Representação baseada em tempo de conclusão		YAMADA; NAKANO, 1997.
Representação baseada por máquina (MB)		DORNDORF; PESCH, 1995.
Representação baseado em regra de prioridade (PR)		DORNDORF; PESCH, 1995.
Representação baseada na relação de pares de *job*s	Codificação Binária	NAKANO; YAMADA, 1997.
Representação baseada em gráfico disjuntivo		TAMAKI; NISHIKAWA, 1992.
Representação por semente dinâmica (SD)		GRASSI; TRIGUIS; PEREIRA, 2014.
Representação por chaves aleatórias (RK)	Codificação real	BEAN1994; GONÇALVES; RESENDE; RUIZ et al., 2015; LALLA-RUIZ et al., 2014; GONÇALVES; RESENDE, 2013

Fonte: Autor

3.5 SOLUÇÕES NÃO FACTÍVEIS E SOLUÇÕES REDUNDANTES

Dentre as representações do Quadro 5, as representações LISTA e SD geram algumas soluções não factíveis, porém não redundantes, no caso da representação OB geram somente soluções factíveis podendo ser redundantes, como, por exemplo, ilustrado a seguir para um problema de sequenciamento com dois jobs (J1, J2) em duas máquinas (M1, M2).

Considerando o problema de *job shop* definido pela sequência tecnológica indicada na Tabela 3, o *job* J1 deve ser processado, obrigatoriamente, primeiro pela máquina M1 e depois pela máquina M2.

Tabela 3 – Sequência Tecnológica

Job	*Máquina (Tempo de processamento)*	
J1	M1 (2)	M2 (1)
J2	M2 (1)	M1 (1)

Fonte: Autor

Nesse caso, existem três sequências de atendimento que definem soluções factíveis. São elas:

- Sequência 1: {1, 2; 2, 1} que indica J1, J2 em M1 e J2, J1 em M2
- Sequência 2: {1, 2; 1, 2} que indica J1, J2 em M1 e J1, J2 em M2
- Sequência 3: {2, 1; 2, 1} que indica J2, J1 em M1 e J2, J1 em M2

Importante observar que a sequência {2, 1; 1, 2} não é factível, pois provoca um bloqueio conhecido como *gridlock*: J1 espera em M1 enquanto J2 espera em M2 (PACHECO; SANTORO, 1999).

A representação por lista de operações (OB) gera cromossomos codificados em listas de números inteiros e diferentes cromossomos são criados pela permutação entre todos os valores dessa lista. Assim, para esse problema, os seguintes cromossomos podem ser gerados: (1, 2, 1, 2), (1, 2, 2, 1), (2, 1, 2, 1), (2, 1, 1, 2), (1, 1, 2, 2) e (2, 2, 1, 1). Cada cromossomo representa uma

sequência de operações dos jobs segundo a ocorrência dos valores, ou seja, o cromossomo (1, 2, 1, 2) define a seguinte sequência de operações:

{1ª operação do job J1, 1ª operação do job 2, 2ª operação do J1, 2ª operação do J2} (1)

Considerando a sequência tecnológica para esse problema (Tabela 3), a sequência de operações em (1) define a seguinte sequência de atendimento nas máquinas: {1, 2; 2, 1} que indica J1, J2 em M1 e J2, J1 em M2. Todas as sequências de atendimento geradas por essa representação são indicadas na Tabela 4.

Tabela 4 – Sequências de atendimento geradas pela representação por lista de operações

Cromossomos	Sequência	Factível?
(1,2,1,2)	{1, 2; 2, 1}	Sim
(1,2,2,1)	{1, 2; 2, 1}	Sim
(2,1,2,1)	{1, 2; 2, 1}	Sim
(2,1,1,2)	{1, 2; 2, 1}	Sim
(1,1,2,2)	{1, 2; 1, 2}	Sim
(2,2,1,1)	{2, 1; 2, 1}	Sim

Fonte: Autor

Todas as sequências apresentadas na Tabela 4 representam soluções factíveis, mas os 4 primeiros cromossomos geram exatamente a mesma sequência.

Por outro lado, na representação binária proposta por Grassi (2014), as soluções são geradas a partir de uma solução inicial factível chamada semente. Assim, a semente pode ser definida como qualquer das três sequências factíveis para esse problema: {1, 2; 2, 1}, {1, 2; 1, 2} e {2, 1; 2, 1}. Os cromossomos nessa representação são codificados como matrizes de números binários e indicam as posições na sequência que deverão sofrer permutação, sempre em uma mesma máquina e com o *job* subsequente, para gerar novas soluções. Todos os cromossomos e as correspondentes sequências de atendimento geradas por

essa representação, obtidos pela permutação da semente {1, 2; 2, 1}, são indicados na Tabela 5.

Essa representação gera uma solução não factível, mas todas as permutações realizadas produzem soluções diferentes e não redundantes. Portanto, investigar essas representações, definir a relação entre operadores de cruzamento e a proporção de soluções factíveis e, em última análise, a melhor forma de representação cromossômica para o problema de sequenciamento da produção em *job shop* é a principal contribuição deste trabalho.

Tabela 5 – Sequências de atendimento geradas pela representação binária

Cromossomos	Sequencia	Factível?
0 0	{1,2;2,1}	Sim
0 1	{1,2;1,2}	Sim
1 1	{2,1;1,2}	Não
1 0	{2,1;2,1}	Sim

Fonte: Autor

3.6 OPERADORES DE CRUZAMENTO PARA CODIFICAÇÃO BINÁRIA

Para cada tipo de representação das soluções têm-se distintos de operadores de cruzamento, pois a operação de cruzamento está intrinsecamente ligada à forma como o cromossomo é codificado. Isto significa que tipos de operadores de cruzamento utilizados na codificação binária, não fazem sentido se aplicados em outras codificações. No contexto da representação binária destacam-se os seguintes tipos de operadores de cruzamento, também conhecidos como canônicos: *One Point Crossover, Uniform Crossover, Even Odd Crossover*.

a) One Point Crossover

Este método é o mais simples, pois escolhe aleatoriamente um único ponto para cruzamento, dividindo o cromossomo em duas partes. Após a divisão, há uma troca de posição dos bits do pai 1 com o do pai 2. O filho 1 terá a parte superior do pai 1 e a parte inferior do pai 2, invertendo essa ordem na composição do filho 2. Um exemplo deste tipo de cruzamento é visualizado na Figura 30.

Figura 30 – Exemplo de cruzamento em um único ponto

Pai 1	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**
Pai 2	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**
Filho 1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**
Filho 2	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

Fonte: Autor

b) Uniform Crossover

O método utilizado pelo operador de cruzamento uniforme faz uso de uma máscara binária gerada aleatoriamente. Adota-se a seguinte regra para a geração do primeiro novo cromossomo (filho 1): coloca-se paralelamente os *bits* da máscara com os *bits* do primeiro e do segundo pai e realiza-se a cópia sequencialmente *bit* a *bit*. Quando o *bit* da máscara for igual a 1 copia-se o valor do *bit* do pai 1 para o filho 1; quando o valor for igual a 0, copia-se o valor do *bit* do pai 2 para o filho1.

A regra é invertida para a geração do segundo cromossomo (filho 2). Essa operação é ilustrada na Figura 31.

Não há grande diferença de desempenho quando se utiliza o operador de cruzamento Uniforme e o de n pontos (n > 1). Para n = 1, tem-se o operador *One Point Crossover.*

Figura 31 – Exemplo de cruzamento uniforme

Máscara	1	1	0	0	1	0	1	1	0
Pai 1	1	0	1	0	1	1	1	0	1
Pai 2	0	0	0	1	1	0	1	0	0
Filho1	1	0	0	1	1	0	1	0	0
Filho 2	0	0	1	0	1	1	1	0	1

Fonte: Autor

c) *Even Odd Crossover*

O método *Even Odd Crossover*, assim como os demais operadores, realiza o cruzamento entre os pais utilizando-se técnicas combinatórias. Seguindo a regra de que para a nova geração, são selecionados os genes com índice par do pai 1 e os genes com índice ímpar do pai 2, ou o contrário, os genes com índice ímpar do pai 1 e índice par do pai 2. Um exemplo deste tipo de cruzamento é visualizado na Figura 32.

Figura 32 – Exemplo de cruzamento por paridade

	I	P	I	P	I	P	I	P	I	P
Pai 1	1	0	1	1	1	0	1	1	0	0
Pai 2	1	1	1	0	0	0	0	1	0	1
Filho 1	1	1	1	0	1	0	1	1	0	1
Filho2	1	0	1	1	0	0	0	1	0	0

Fonte: Autor

3.7 OPERADORES DE CRUZAMENTO PARA CODIFICAÇÃO INTEIRA.

Como já foi comentado, para cada tipo de representação ou codificação são necessários operadores de cruzamento especializados. No contexto da representação por lista de preferências, ou representações que utilizem a codificação inteira destacam-se os seguintes tipos de operadores de cruzamento: PMX, OX, CX, LOX e PPX.

a) PMX – *Partially Mapped Crossover*

O operador PMX foi proposto por Goldberg e Lingle (MICHALEWICZ, 1996), baseia-se em trocar uma sequência de genes limitada entre dois pontos de corte escolhidos aleatoriamente, entre os pais, para se formar uma nova geração de cromossomos. Após a troca, o preenchimento dos alelos faltantes, segue a sequência do pai original não repetindo alelos, para não se gerar um cromossomo inviável, esse tipo de cruzamento é visualizado na Figura 33.

Figura 33 – Exemplo de cruzamento PMX

$$\boldsymbol{p1}\colon \mathbf{1\,2\,4} \uparrow \mathbf{7\,6\,5} \uparrow \mathbf{3} \rightarrow \boldsymbol{x\,x\,x} \uparrow \mathbf{1\,7\,5} \uparrow \boldsymbol{x} \rightarrow \boldsymbol{f1}\colon \mathbf{2\,4\,6} \uparrow \mathbf{1\,7\,5} \uparrow \mathbf{3}$$

$$\boldsymbol{p2}\colon \mathbf{2\,3\,4} \uparrow \mathbf{1\,7\,5} \uparrow \mathbf{6} \rightarrow \boldsymbol{y\,y\,y} \uparrow \mathbf{7\,6\,5} \uparrow \boldsymbol{y} \rightarrow \boldsymbol{f2}\colon \mathbf{2\,3\,4} \uparrow \mathbf{7\,6\,5} \uparrow \mathbf{1}$$

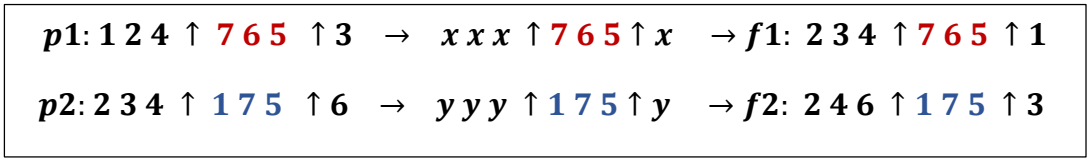
Fonte: Autor

b) OX – *Order Crossover*

O operador OX foi proposto por Davis (MICHALEWICZ, 1996), análogo ao PMX, com a diferença que os genes limitados entre os dois cortes não são trocados entre os pais, mas agora são herdados integralmente na nova geração. O preenchimento dos alelos faltantes segue a sequência do outro pai não

repetindo alelos, para não se gerar um cromossomo inviável, esse tipo de cruzamento é visualizado na Figura 34.

Figura 34 – Exemplo de cruzamento OX

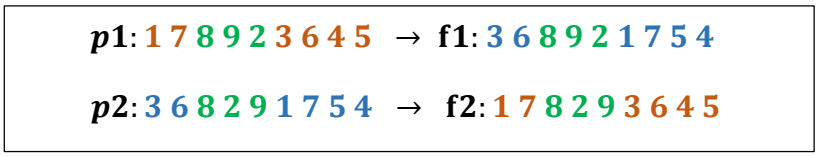


Fonte: Autor

c) CX – *Cycle Crossover*

O operador CX foi proposto por Oliver (MICHALEWICZ, 1996), a lógica deste operador consiste em buscar entre os cromossomos dos pais conjuntos de genes semelhantes com a mesma posição e mantê-los na futura geração, os genes faltantes são preenchidos com os genes do outro pai, esse tipo de cruzamento é visualizado na Figura 35.

Figura 35 – Exemplo de cruzamento CX



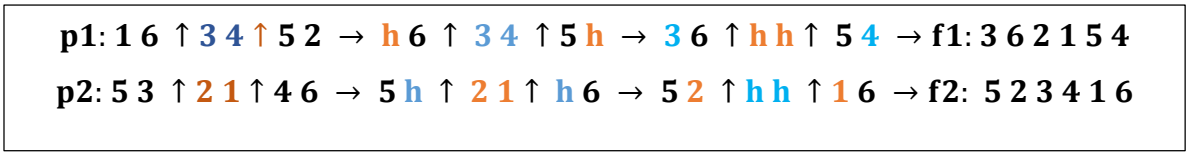
Fonte: Autor

d) LOX – *Linear Order Crossover*

O operador LOX foi proposto por Falkenauer e Bouffouix (CROCE; TADEI; VOLTA, 1995). Neste tipo de operador existem três estágios: primeiro gera-se os pontos de corte aleatoriamente de mesmo tamanho e posição e escolhe-se a

cadeia de genes; segundo marca-se os alelos idênticos à cadeia do outro pai; em terceiro move-se estes alelos marcados para a posição da cadeia e após isso se faz a troca entre os pais. Esse tipo de cruzamento é visualizado na Figura 36.

Figura 36 – Exemplo de cruzamento LOX



Fonte: Autor

e) PPX - *Precedence Preservative Crossover*

O operador PPX foi proposto por Mattfeld e Bierwirth (2004), utiliza uma máscara binária para definir como o gene será transmitido para a próxima geração. Quando o valor da máscara for “0”, herda o gene do pai inicial, caso contrário herda do outro pai, por trabalhar-se com números inteiros deve-se observa se a combinação não é viável, caso isso aconteça faz-se o sequenciamento do próximo gene, para viabilizar o filho, este procedimento pode ser visualizado na Figura 37.

Figura 37 – Exemplo de cruzamento PPX

Máscara: 0 1 1 0 1 0
p1: 1 2 3 4 6 5 → f1: 1 6 3 2 4 5 **p2: 6 3 1 4 2 5 → f2: 6 1 2 4 3 5**

Fonte: Autor

3.8 OPERADORES DE SELEÇÃO

O operador de seleção visa, como na genética, garantir que os indivíduos mais aptos sejam mais frequentemente selecionados, para perpetuar suas boas características. Entretanto, esse processo deve garantir também que mesmo indivíduos menos aptos sejam selecionados, visando manter uma diversidade da população e evitar convergência prematura. Segundo Liden (2012), as seguintes premissas devem ser respeitadas: o método que imita a seleção dos pais deve simular ao máximo a realidade do mecanismo de seleção natural, ou seja, pais mais aptos, capazes geram mais filhos; em contrapartida pais menos aptos também podem gerar descendentes, mas com menor frequência.

A justificativa em manter na nova população uma minoria de indivíduos com avaliação ruim se deve a possibilidade de existir características genéticas que sejam favoráveis à criação de um indivíduo ótimo às quais, podem não estar presentes em nenhum outro.

Principais métodos de seleção utilizados são:

- Método da Roleta

  O AG clássico com codificação binária utiliza o método da roleta (*roulette wheel*), esse método consiste em atribuir uma probabilidade de passar para próxima geração proporcional à aptidão (*fitness*), em relação a somatória de todas as aptidões atribuídas. Indivíduos com maior aptidão possuirão sempre probabilidade maior de se manterem na próxima geração (MICHALEWICZ, 1996).

  Este método não garante que o melhor indivíduo não seja descartado. Em alguns casos opta-se por permitir que uma porcentagem dos melhores indivíduos seja obrigatoriamente mantida na população seguinte, método conhecido como seleção elitista (FOGEL, 1994; MICHALEWICZ, 1996).

- Método do Torneio

Basicamente consiste em simular um torneio entre indivíduos. A seleção é feita em função do número de vitórias em q competições contra oponentes aleatórios da população, ganhando o que apresentar o maior *fitness*. Segundo a literatura é um dos métodos mais sofisticados. Uma descrição mais detalhada desse operador pode ser encontrada em Linden (2012).

- Seleção Local

Este método funciona com a noção de busca local em torno da vizinhança de determinada solução já selecionada por qualquer outro método já citado anteriormente.

- Método De Amostragem Estocástica Uniforme

Neste método todos os indivíduos são ordenados em uma reta. Cada indivíduo possui um tamanho na reta equivalente à sua avaliação, sendo que a soma total de todos os seguimentos será igual a 1. É escolhido um valor de tamanho n, para ir saltando e escolhendo na sequência os valores, claramente os indivíduos que possuírem um tamanho maior no segmento terão maior probabilidade de serem selecionados.

- Seleção por *ranking*

Este método é conhecido por evitar a convergência prematura e de que um indivíduo acabe dominando no processo de seleção. Seu

funcionamento segue os seguintes passos: ordenar todos os elementos de acordo com a sua função de avaliação e usar este “ranking” como base da seleção; feito este mapeamento passa-se a utilizar qualquer outro método, este processo possui um custo computacional alto, segundo Linden (2012).

- Seleção Truncada

Este método primeiramente posiciona cada indivíduo em ordem decrescente de acordo com a sua avaliação, ponderando valores entre 1 e 0, em seguida é escolhido um valor para corte. Apenas os valores que estiverem entre 1 e o valor de corte permanecem para a próxima população. Os valores de corte geralmente estão entre 10% e 50%.
Este método causa uma convergência mais veloz e pouca diversidade entre os indivíduos das próximas gerações, a ordenação gera um custo computacional.

3.9 OPERADORES DE MUTAÇÃO

Os operadores de mutação são utilizados para garantir maior diversidade nas próximas gerações e também para que não ocorra uma convergência muito rápida em torno de ótimos locais.

Da mesma forma que na genética a taxa de mutação não deve ser elevada, caso contrário acabar-se-ia degradando os métodos de seleção, tornando o método apenas aleatório e sem nenhum elitismo. Os valores atribuídos estão entre 0,5% até 5%.

Os operadores mais utilizados são: *swap mutator* e *flip bit.*

Swap mutator é mais usual em representações por permutação, consiste em permutar posições dentro de um cromossomo aleatoriamente, em determinada taxa.

Flip bit é utilizado em representações binárias e consiste em trocar o valor de um ou mais bits escolhidos aleatoriamente dentro do cromossomo, também utiliza a taxa para selecionar a quantidade de bit.

O próximo capitulo esclarece as características da pesquisa, bem como foram definidos os experimentos e os parâmetros adotados.

4 MATERIAIS E MÉTODOS

4.1 CARACTERIZAÇÃO DA PESQUISA

A presente pesquisa visou realizar um estudo comparativo, de acordo com o objetivo, as características que envolvem conhecer a melhor representação cromossômica no AG, aplicados na solução de JSSP. Assim sendo, adotando a classificação apresentada por Gil (2008), configura-se como uma pesquisa exploratória e explicativa.

O trabalho de pesquisa no seu desenvolvimento caracterizou-se por sua abordagem quantitativa e experimental, pois segundo Martins (2012) trata-se de uma pesquisa na qual as variáveis são avaliadas, medidas por meio de valores quantificáveis e, ainda se baseando em Severino (2007), lida com elementos concretos, em condições de observação e manipulação experimental.

Segundo Morabito Neto e Pureza (2012), as pesquisas quantitativas dentro da gestão de produção e operações podem ser divididas de acordo com a orientação do pesquisador e a fonte de dados. Nesse caso, a orientação é racional e dedutiva, pois segundo Marconi e Lakatos (2010) a dedução trata-se de um processo mental no qual se procura estabelecer devidas previsões e explicações para determinados fatos ou fenômenos, o que se aplica a este trabalho.

Como passo inicial deste trabalho, foi executado uma pesquisa exploratória, através da pesquisa bibliográfica horizontal que serviu para proporcionar maior familiaridade com o problema, dar rumo à definição do problema de pesquisa, bem como identificar lacunas na literatura (FLEURY, 2012). Como passo seguinte, e seguindo ainda orientação de Fleury (2012), efetuar uma revisão vertical para aprofundar o conhecimento sobre o tema e balizar seu desenvolvimento.

4.2 BIBLIOTECA E PARÂMETROS DO AG

A implementação das versões do AG tratadas neste trabalho foi toda desenvolvida por meio da GAlib, que é uma biblioteca de Algoritmos Genéticos, escrita em C++ por Matthew Wall, do *Massachusetts Institute of Technology* (GAlib, 2012), acessível em http://lancet.mit.edu/ga/GAlib.html.

Dois conjuntos de experimentos foram realizados: o primeiro conjunto de testes considerou o uso de diferentes operadores e taxas de cruzamento na representação SD. Nesse caso, os experimentos foram planejados visando avaliar os efeitos dos operadores e da taxa de cruzamento nos resultados de *makespan* e da proporção de soluções factíveis. Os resultados do SD foram comparados com o trabalho de Abdelmaguid (2010), que utilizou o AG com seis diferentes representações: OB, RK, LISTA, PR, MB e JB. Todas estas representações não utilizam a codificação binária, sendo este o motivo principal da escolha, e com isso efetuar o estudo comparativo da eficiência da representação SD, visto esta utilizar a codificação binária.

No Quadro 6, é possível visualizar um resumo dos parâmetros utilizados no AG para as avaliações propostas no primeiro conjunto de experimentos. As escolhas dos valores foram feitas com base na literatura pertinente (YAMADA, 2003; MITCHELL, 1997).

O segundo conjunto de testes compara as representações SD, baseado em operação (OB) e lista de preferências (LISTA). Escolheu-se a representação OB, pelo motivo da mesma não gerar soluções não factíveis, ou seja, somente gera soluções viáveis (DAVIS,1985) comparando-a com a SD que gera uma proporção muito pequena de não factíveis (GRASSI, 2014) em torno de 5% e também comparando com a representação LISTA por gerar uma quantidade excessiva de soluções não factíveis e ser uma das mais utilizadas (GEN et al. 1994; KUBOTA, 1995; FAN et al 1993).

Quadro 6 – Parâmetros do AG adotados no primeiro experimento

Parâmetro	**Valor Adotado**
Representação do Cromossomo	Binária (SD)
Seleção	*Steady State*
Taxa de Substituição da População	90%
Tamanho da População	10
Cruzamento	*One Point Crossover (OP), Uniform Crossover (UN), Even Odd Crossover (ED)*
Taxa de Cruzamento	30%, 50%, 70% e 90%
Mutação	Inversão de bit
Taxa de Mutação	1%
Número de Gerações	≥10000 (25 internas e 400 externas)
Critério de Parada	25 gerações internas sem convergência + 400 gerações externas

Fonte: O Autor

Para cada tipo de representação abordada, executou-se 10 simulações, para cada exemplar. Foram testadas as seguintes representações, nessa ordem:

1. Semente dinâmica com 20 laços externos denotada por SD20;
2. Semente dinâmica com 10 laços externos (SD10);
3. Representação baseada em operações (OB), e
4. Representação por Lista de preferências (LISTA).

Nos experimentos, a população inicial gerada aleatoriamente para a representação SD foi armazenada, em cada uma das 10 execuções, e usada como entrada nas demais representações testadas posteriormente. Essa prerrogativa foi adotada para garantir condições iniciais semelhantes, no momento de comparar e analisar o desempenho de cada representação proposta.

Os parâmetros usados no segundo conjunto de experimentos são apresentados no Quadro 7, as escolhas dos valores foram feitas com base na literatura pertinente (YAMADA, 2003; MITCHELL, 1997).

No caso do SD, diferentes valores de laços externos são testados para avaliar a influência desse fator no desempenho da abordagem e definir a melhor configuração em relação ao custo computacional.

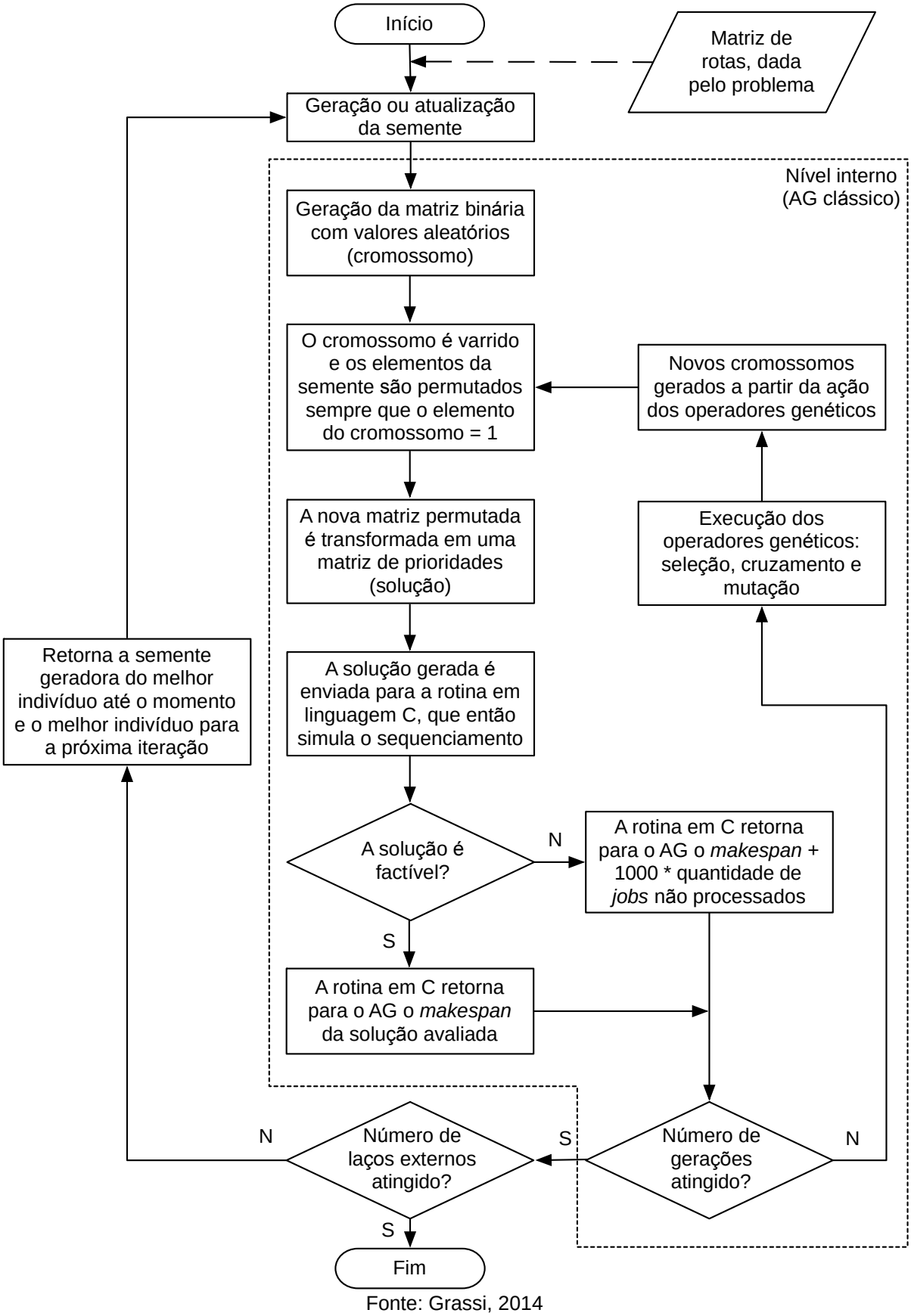
Quadro 7 – Parâmetros do AG adotados para SD, OB e LISTA

Parâmetro	**SD**	**OB**	**LISTA**
Representação do Cromossomo	Binária	Inteira	Inteira
Seleção	*Steady State*	*Steady State*	*Steady State*
Taxa de Substituição da População	80%	80%	80%
Tamanho da População	50	50	50
Cruzamento	*One Point*	*PMX*	*PMX*
Taxa de Cruzamento	90%	90%	90%
Mutação	Inversão de bit	Inversão de bit	Inversão de bit
Taxa de Mutação	1%	1%	1%
Número de Gerações	**(Número de Jobs x quantidade de Máquinas) / 2** gerações internas	**500**	**500**
Critério de Parada	Número Gerações	Número Gerações	Número Gerações

Fonte: O Autor

Adicionalmente, um nível externo é executado a fim de atualizar a semente dinamicamente. Após a execução de um determinado número de gerações do AG clássico no nível interno, a melhor solução obtida e a semente permutada correspondente são utilizadas em uma nova iteração interna do AG. Esse procedimento é ilustrado na Figura 38.

Figura 38 – Fluxograma da metodologia experimental proposta



Fonte: Grassi, 2014

4.3 OS PROBLEMAS DE *JOB SHOP* TESTADOS

A família ou instâncias de exemplares *job shop*, consiste num conjunto de exemplares teóricos desenvolvidos por pesquisadores no intuito de embasar e dar subsidio na pesquisa de problemas de sequenciamento. Neste trabalho são utilizadas as instâncias de Lawrence (1984).

Lawrence desenvolveu os modelos teóricos conhecidos como LA01 a LA40, os quais são exemplares *job shop* para problemas de vários tamanhos e diferentes complexidades, conforme visualizados na Tabela 6. Os primeiros 20 problemas são utilizados neste trabalho, variando do LA01 até o LA05 para testes da influência dos operadores de cruzamento e do LA01 até LA20 para as demais comparações.

Tabela 6 – Família de exemplares LA com número de *job*s e máquinas

Instâncias	Número de *job*s	Número de Máquinas
LA01 – LA05	10	5
LA06 – LA10	15	5
LA11 – LA15	20	5
LA16 – LA20	10	10
LA21 – LA25	15	10
LA26 – LA30	20	10
LA31 – LA35	30	10
LA36 – LA40	15	15

Fonte: Autor

4.4 RECURSOS COMPUTACIONAIS UTILIZADOS

Os recursos computacionais utilizados nos experimentos citados, foram distintos, ou seja, para o primeiro conjunto de experimentos utilizou-se um notebook ASUS modelo K45VM com processador "CORE i7”, 2,3 GHz, memória de 8 Gbytes, com sistema operacional Windows 7 Home Basic 64bits.

Para o segundo conjunto de experimentos utilizou um *Ultrabook* da HP modelo Elite *BookFolio* 9470m com processador “CORE™ i5™ vPro™”, frequência do *clock* 1,80 GHz 2,4GHz, memória de 8Gbytes, o sistema operacional foi Windows 7 profissional 64bits.

Nos experimentos de Abdelmaguid (2010) foi utilizado um computador pessoal com processador Intel Pentium™ CORE 2 DUO™ com frequência de *clock* de 2,67GHz e 512Mbytes de memória, não há referência sobre qual o tipo do sistema operacional.

4.5 FERRAMENTAS ESTATÍSTICAS UTILIZADAS

Para as devidas análises foram utilizadas ferramentas estatísticas, no primeiro conjunto de experimentos utilizou-se a correlação (R), medida relativa que avalia a relação entre duas variáveis, verificando se uma tem dependência da outra, quando mais relacionadas o valor de R tende a 1, também se utilizou análise de variância entre várias médias, chamada de ANOVA, no caso testes e hipótese, relacionados com a média amostral, para avaliar a interação de vários grupos. Na utilização da ANOVA os valores F e $F_{crítico}$ são cruciais para se aceitar ou rejeitar a hipótese inicial, em relação a hipótese alternativa, tudo isso com base na somatória da variância (SQ) e na média da variância (MQ).

Para o segundo conjunto de experimentos fez-se uso das médias amostrais, valores mínimos e máximos de cada grupo, bem como o desvio percentual, e comparações entre estes valores.

Para todos os cálculos utilizou-se os recursos de Análise de dados do Excel Office.

5 RESULTADOS E DISCUSSÕES

.

Os resultados dos experimentos são apresentados e discutidos neste capítulo. Adotou-se uma divisão em subseções com o intuito de facilitar a análise de cada um dos objetivos definidos no capítulo I.

5.1 INFLUÊNCIA DOS OPERADORES DE CRUZAMENTO NO SD

Os resultados apresentados na Tabela 7 visam avaliar os efeitos dos operadores de cruzamento binários, no SD, aplicado ao JSSP em relação à qualidade da solução e a proporção de soluções factíveis geradas no processo de otimização. A qualidade da solução é medida, nesse caso, pelos valores de *makespan* obtidos com cada uma das representações. Para verificar a influência dos operadores foram utilizados os operadores canônicos: *One Point Crossove*r (OP), *Uniform Crossover* (UN), *Even Odd Crossover* (ED). Para cada tipo operador de cruzamento, também se testou a taxa de cruzamento com os seguintes valores: 0,3; 0,5; 0,7; 0,9.

Tabela 7 – Comparação das melhores soluções por representação

Problema	Número de Operações	*Makespan* Ótimo Conhecido	*OB*	*RK*	*LISTA*	*PR*	*MB*	*JB*	SD
LA01	50	666	666	666	675	671	666	700	**666**
LA02	50	655	676	686	715	675	684	718	**655**
LA03	50	597	631	637	669	650	625	645	617
LA04	50	590	607	614	633	629	**590**	675	595
LA05	50	593	593	593	593	593	593	605	**593**

Fonte: Adaptado de Abdelmaguid, 2010

Os valores na coluna do SD apresentados em negrito significam que o algoritmo atingiu o *makespan* ótimo conhecido (abreviado para MKP*).

Para os problemas testados, o SD apresentou resultados iguais ou melhores em relação a todos os *jobs shops* testados, exceção feita ao problema LA04 no qual a abordagem MB encontrou o ótimo conhecido e o SD apresentou um desvio de 0,8% em relação ao ótimo. Vale destacar, no entanto, que a representação MB leva uma média de 550 segundos para resolver problemas com 50 operações, contra cerca de 35 segundos do SD.

Os operadores e as taxas de cruzamento foram variados conforme definido no Quadro 6 da seção 4.2, pg. 86. Os resultados são apresentados nas Tabelas 8 - 12. Nas Tabelas, além do *makespan* e da proporção de soluções factíveis, é também apresentado o número total de avaliações em cada caso. Valores em negrito representam que a melhor solução conhecida foi obtida, e quando o símbolo † aparecer, indica que a execução foi interrompida inesperadamente. Como esperado, o número total de avaliações aumenta com a taxa de cruzamento, pois números maiores de novos indivíduos criados devem ser avaliados.

De fato, considerando o problema LA01, o número total de avaliações é a resposta observada que mais sofre influência do fator taxa de cruzamento. Observa-se também uma influência da taxa de cruzamento, ainda que menor, sobre a proporção de soluções factíveis, como pode ser notado na Figura 39 observando que, nesses dois casos, tem-se $F > F_{crítico}$ na análise de variância.

Quadro 8 – Correlação entre as variáveis taxa de cruzamento e proporção de factíveis

	LA01	**LA02**	**LA03**	**LA04**	**LA05**
Correlação	0,986269	0,944006	0,802255	-0,301291	0,9696858

Fonte: Autor

A influência da taxa de cruzamento sobre a proporção de soluções factíveis é especialmente importante, pois é o reflexo de uma correlação positiva entre essas duas variáveis, analisando os cinco exemplares, nota-se para os casos LA01, LA02, LA03 e LA05 que o coeficiente de correlação R fica entre 0,80 e 0,98 como visualizado no Quadro 8.

Figura 39 – Análise dos efeitos dos fatores sobre as respostas observadas para o problema LA01: (a) *makespan*, (b) proporção de soluções factíveis e (c) número de avaliações

Fonte da variação	*SQ*	*gl*	*MQ*	*F*	*valor-P*	*F crítico*
Linhas	51,16667	2	25,58333	0,259803	0,779453	5,14325285
Colunas	776,6667	3	258,8889	2,629055	0,144785	4,757062664
Erro	590,8333	6	98,47222			
Total	1418,667	11				

(a)

Fonte da variação	*SQ*	*gl*	*MQ*	*F*	*valor-P*	*F crítico*
Operadores	2,82E-07	1	2,82E-07	0,050193	0,843532	18,51282051
Taxa de cruzamento	0,00032	2	0,00016	28,52658	0,033868	19
Erro	1,12E-05	2	5,61E-06			
Total	0,000332	5				

(b)

Fonte da variação	*SQ*	*gl*	*MQ*	*F*	*valor-P*	*F crítico*
Operadores	14406	1	14406	0,608272	0,517083	18,51282051
Taxa de cruzamento	5,06E+08	2	2,53E+08	10673,13	9,37E-05	19
Erro	47367	2	23683,5			
Total	5,06E+08	5				

(c)

Fonte: O Autor

Isso significa que quanto maior a taxa de cruzamento mais soluções factíveis são geradas. Trata-se de um resultado muito interessante que mostra a vantagem da abordagem SD ao usar os operadores de cruzamento tradicionais.

Tabela 8 – Resultados para problema LA01

Operador	Taxa	*Makespan*	Factíveis %	Avaliações
ED	0,3	667	93,23	52027
ED	0,5	667	94,26	62811
ED	0,7	696	94,72	74056
ED	0,9	**666**	95,61	84922
OP	0,3	667	93,31	52003
OP	0,5	667	94,17	62782
OP	0,7	696	94,84	74159
OP	0,9	667	95,84	85151
UN	0,3	†	†	†
UN	0,5	678	93,83	62871
UN	0,7	667	95,16	74044
UN	0,9	667	95,73	85471

Fonte: O Autor

Tabela 9 – Resultados para problema LA02

Operador	Taxa	*Makespan*	Factíveis %	Avaliações
ED	0,3	705	92,69	52149
ED	0,5	667	94,22	63216
ED	0,7	670	94,64	74410
ED	0,9	670	94,64	74410
OP	0,3	660	93,18	52315
OP	0,5	667	93,83	62994
OP	0,7	672	94,78	74198
OP	0,9	**655**	95,65	84957
UN	0,3	660	93,14	52577
UN	0,5	710	93,72	63081
UN	0,7	667	94,90	74121
UN	0,9	672	95,48	85145

Fonte: O Autor

Por fim, mas não menos importante, foi possível observar em todos os casos (exceto no problema LA05) uma baixa correlação negativa entre a proporção de soluções factíveis geradas e o valor de *makespan* (R em torno de -0,5). Isso indica que, em geral, a geração de um maior número de soluções viáveis pode conduzir a melhores valores de *makespan*, como é de se esperar.

Tabela 10 – Resultados para problema LA03

Operador	Taxa	*Makespan*	Factíveis %	Avaliações
ED	0,3	617	93,03	52162
ED	0,5	617	94,07	63223
ED	0,7	617	94,76	73965
ED	0,9	617	95,75	85159
OP	0,3	617	93,34	52235
OP	0,5	617	94,29	63698
OP	0,7	617	95,06	74250
OP	0,9	617	95,35	85622
UN	0,3	617	93,05	52231
UN	0,5	632	92,68	63125
UN	0,7	639	93,96	74098
UN	0,9	651	94,13	85083

Fonte: O Autor

Tabela 11 – Resultados para problema LA04

Operador	Taxa	*Makespan*	Factíveis %	Avaliações
ED	0,3	613	92,46	91590
ED	0,5	607	93,43	91383
ED	0,7	607	91,93	91572
ED	0,9	607	92,12	91851
OP	0,3	607	92,24	91374
OP	0,5	607	92,20	91140
OP	0,7	602	91,59	91365
OP	0,9	607	92,93	92130
UN	0,3	595	94,46	91140
UN	0,5	607	93,33	91374
UN	0,7	595	93,45	91572
UN	0,9	607	92,67	91041

Fonte: O Autor

Tabela 12 – Resultados para problema LA05

Operador	Taxa	*Makespan*	Factíveis %	Avaliações
ED	0,3	**593**	92,51	51620
ED	0,5	**593**	94,10	62603
ED	0,7	**593**	95,35	73610
ED	0,9	**593**	95,55	84516
OP	0,3	**593**	92,51	51620
OP	0,5	**593**	94,01	62724
OP	0,7	**593**	95,12	73482
OP	0,9	**593**	95,70	84425
UN	0,3	**593**	93,08	51937
UN	0,5	**593**	93,84	62603
UN	0,7	**593**	95,22	73610
UN	0,9	**593**	95,66	84516

Fonte: O Autor

5.2 INFLUÊNCIA DOS LAÇOS EXTERNOS NA REPRESENTAÇÃO SD

A representação com Semente Dinâmica foi implementada primeiramente com 20 laços externos (SD20), para comparação e verificar a eficiência da técnica dos laços externos, experimentou-se uma versão com apenas 10 laços externos (SD10). Nas Tabelas 13 e 14 visualiza-se os valores mínimos, máximos e médios encontrados em cada um dos problemas testados para as representações SD20 e SD10, os valores em negrito indicam que o AG encontrou o valor de *makespan* ótimo conhecido, quando os valores de máximo, mínimo e médio forem iguais, significa que nas dez rodadas os valores foram idênticos.

São apresentados na sequência os gráficos gerados a partir das Tabelas 13 e 14, nas Figuras 40 e 41, para uma visualização de como foi a distribuição das dez rodadas de cada experimento.

Tabela 13 – Representação SD20 – Valores de *makespan* obtidos

Problema	Mínimo	Média	Máximo
LA01	**666**	674,9	692
LA02	667	692,1	734
LA03	615	632,1	674
LA04	597	624,1	632
LA05	**593**	593	593
LA06	**926**	926	926
LA07	890	894,3	916
LA08	**863**	863	863
LA09	**951**	951	951
LA10	**958**	958	958
LA11	**1222**	1222	1222
LA12	**1039**	1039	1039
LA13	**1150**	1150	1150
LA14	**1292**	1292	1292
LA15	1207	1219,5	1255
LA16	987	1016,9	1035
LA17	793	811,7	848
LA18	861	896,6	932
LA19	876	898,7	911
LA20	914	929,1	958

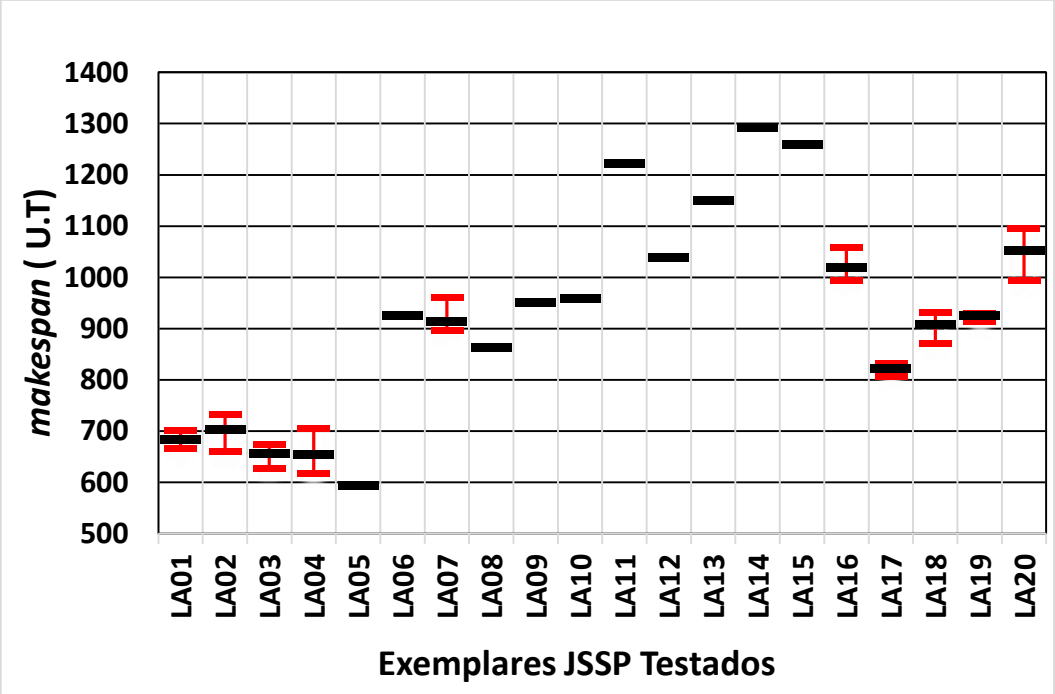
Fonte: O Autor

Tabela 14 – Representação SD10 – Valores de *makespan* obtidos

Problema	Mínimo	Média	Máximo
LA01	681	726,5	765
LA02	734	754,7	813
LA03	634	684,6	711
LA04	650	716,9	761
LA05	**593**	593,6	596
LA06	**926**	929,5	944
LA07	953	978,2	1014
LA08	**863**	900,5	940
LA09	**951**	956,4	960
LA10	**958**	958	958
LA11	**1222**	1222	1222
LA12	**1039**	1044	1060
LA13	**1150**	1150	1150
LA14	**1292**	1292	1292
LA15	1370	1404	1439
LA16	1051	1106,9	1135
LA17	848	866,8	900
LA18	932	951,7	988
LA19	970	996,1	1025
LA20	1160	1172,9	1199

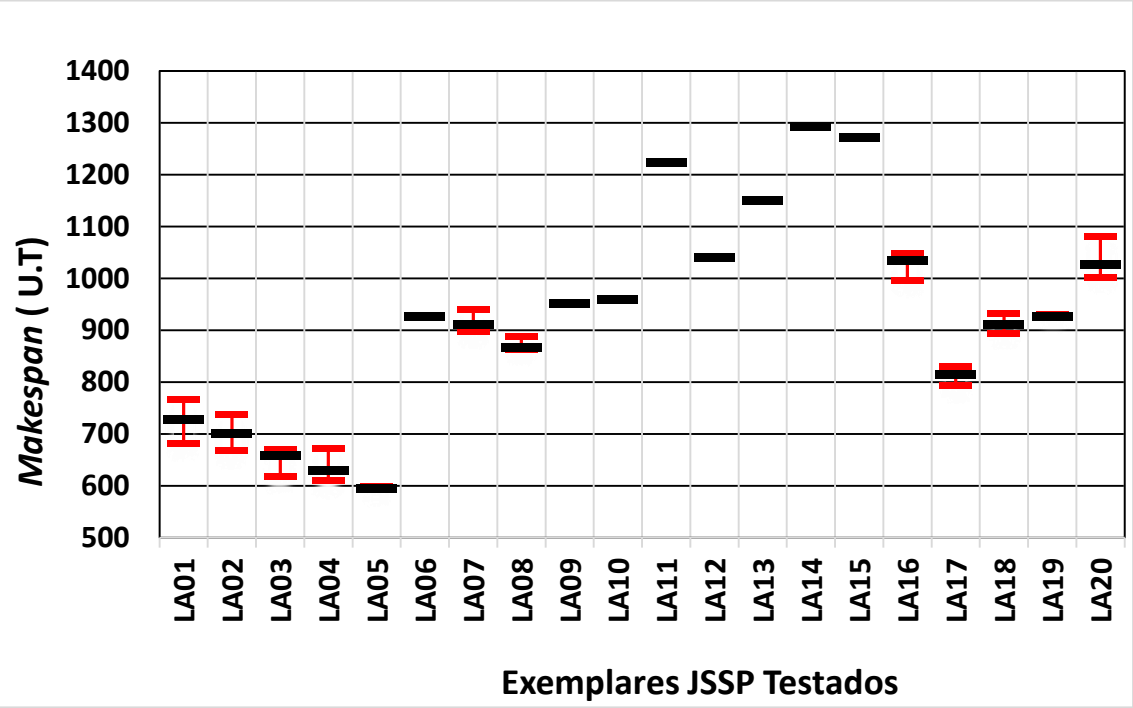
Fonte: O Autor

Figura 40 – Representação SD20 – Valores mínimos, médios e máximos do *makespan* obtidos



Fonte: O Autor

Figura 41 – Representação SD10 – Valores mínimos, médios e máximos do *makespan* obtidos



Fonte: O Autor

Comparando-se a representação SD20 com seu par SD10, verifica-se que o desempenho de ambas é muito similar, apesar da representação SD20 gerar uma quantidade maior de soluções, a proporção de soluções factíveis se manteve equivalente à representação SD10 e esses dados são apresentados na Tabela 15.

Tabela 15 – Quantidades de soluções e proporção de factíveis

Problema	SD20 %	SD10 %	LISTA %	OB %
LA01	95,92	96,17	66,80	100,00
LA02	95,92	96,30	66,58	100,00
LA03	95,62	95,98	66,55	100,00
LA04	95,43	96,31	66,47	100,00
LA05	95,64	95,93	67,75	100,00
LA06	98,37	98,17	52,83	100,00
LA07	98,12	98,07	53,21	100,00
LA08	98,29	98,73	52,66	100,00
LA09	98,39	98,46	53,64	100,00

Fonte: O Autor

Tabela 15 – Quantidades de soluções e proporção de factíveis (cont.)

Problema	SD20 %	SD10 %	LISTA %	OB %
LA10	98,49	98,87	52,55	100,00
LA11	99,49	99,52	42,27	100,00
LA12	99,14	99,15	41,64	100,00
LA13	99,59	99,57	42,24	100,00
LA14	98,97	98,99	41,68	100,00
LA15	99,09	99,26	42,17	100,00
LA16	96,66	96,77	40,50	100,00
LA17	96,47	96,58	40,86	100,00
LA18	95,56	95,73	40,65	100,00
LA19	96,63	96,50	40,59	100,00
LA20	95,12	95,68	40,71	100,00

Fonte: O Autor

Os percentuais de desvio em relação, são apresentados na Tabela 16 e 17 para as representações SD20 e SD10, os exemplares onde não há desvio não são apresentados nas Tabelas.

Tabela 16 – Representação SD20 – Desvio percentual em relação ao MKP*

Problema	MKP*	Mínimo %	Médio %	Máximo %
LA01	666	0,00	2,51	5,41
LA02	655	0,76	7,22	11,76
LA03	597	5,19	9,77	12,90
LA04	590	4,75	10,95	19,49
LA07	890	0,79	2,73	7,98
LA15	1207	4,31	4,31	4,31
LA16	945	5,29	7,93	11,96
LA17	784	2,81	4,83	6,25
LA18	848	2,71	7,06	9,91
LA19	842	8,55	9,94	10,33
LA20	902	10,20	16,69	21,40

Fonte: O Autor

Tabela 17 – Representação SD10 – Desvio percentual em relação ao MKP*

Problema	MKP*	Mínimo %	Médio %	Máximo %
LA01	666	2,25	9,08	14,86
LA02	655	1,83	6,96	12,37
LA03	597	3,35	10,05	12,06
LA04	590	3,22	6,68	13,73
LA05	593	0,00	0,10	1,01

Fonte: O Autor

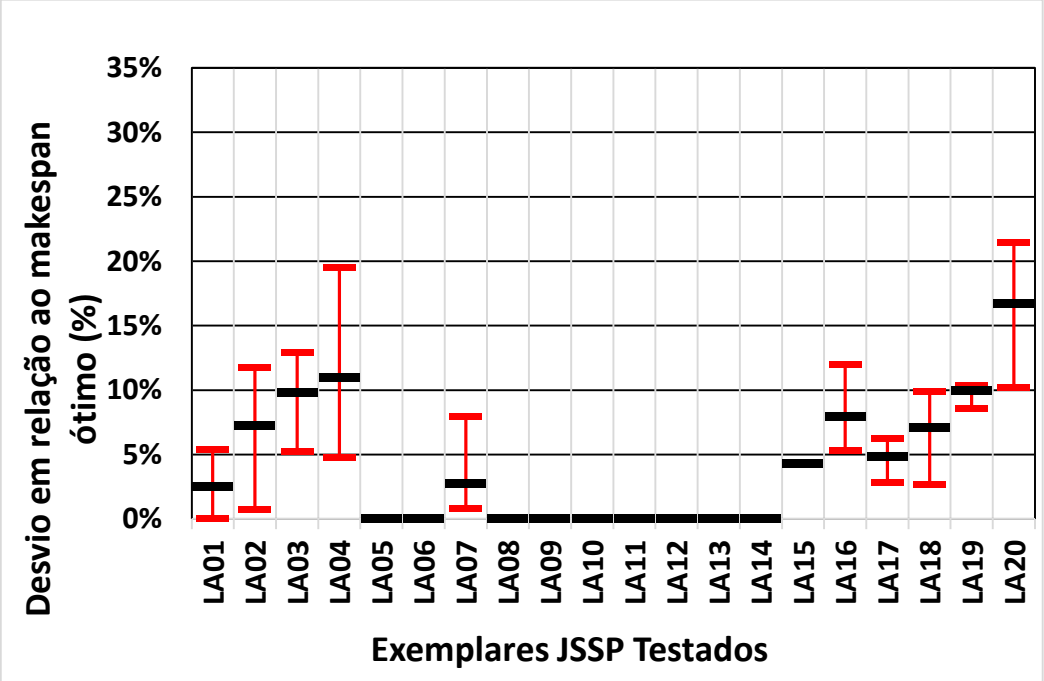
Tabela 17 – Representação SD10 – Desvio percentual em relação ao MKP* (cont.)

Problema	MKP*	Mínimo %	Médio %	Máximo %
LA07	890	0,79	2,73	7,98
LA08	863	0,00	0,29	2,90
LA15	1207	5,30	5,30	5,30
LA16	945	5,29	9,46	10,79
LA17	784	1,15	3,94	5,87
LA18	848	5,42	7,39	9,91
LA19	842	9,86	10,01	10,45
LA20	902	11,09	13,81	19,84

Fonte: O Autor

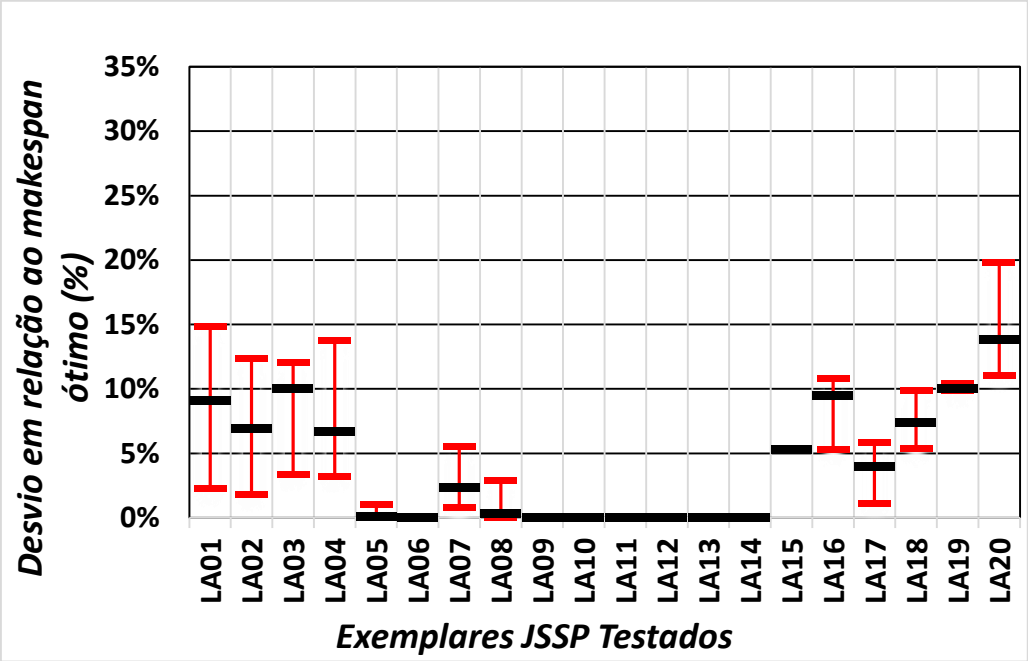
São apresentados na sequência os gráficos gerados a partir das Tabelas 16 e 17, nas Figuras 42 e 43, para uma visualização de como foi a distribuição das dez rodadas de cada experimento.

Figura 42 – Representação SD20 – Valores mínimos, médios e máximos de desvio obtidos



Fonte: O Autor

Figura 43 – Representação SD10 – Valores mínimos, médios e máximos do desvio obtidos



Fonte: O Autor

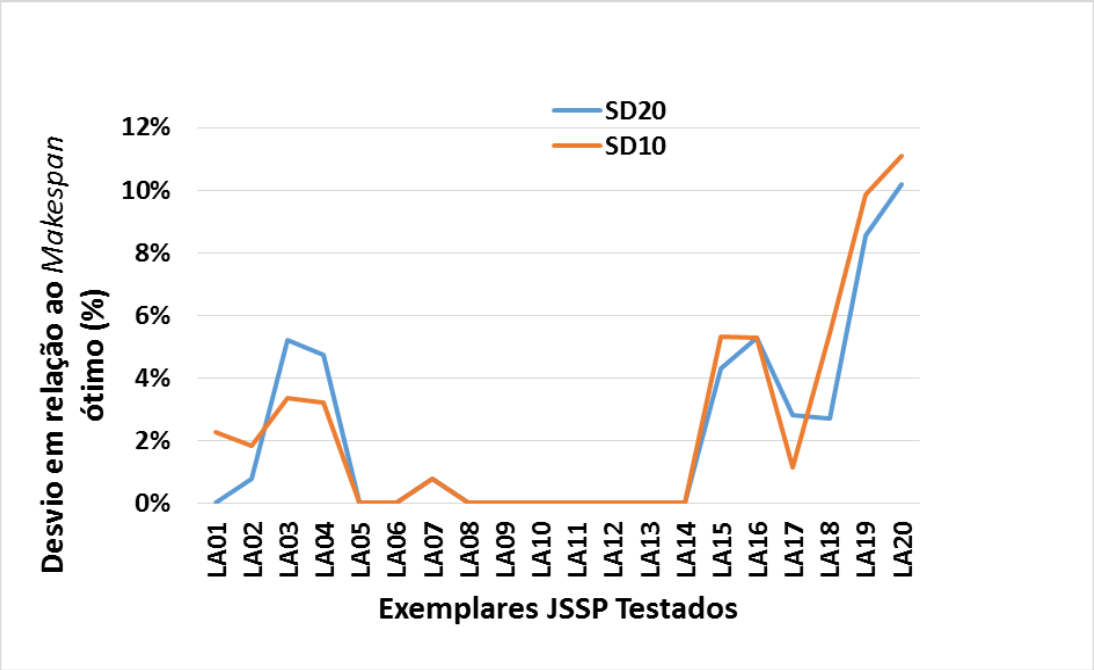
Na Figura 44 pode-se observar o gráfico do desvio percentual do mínimo *makespan* achado em cada representação em relação ao MKP*, baseado nas Tabelas 16 e 17, percebe-se que a representação SD20 gera praticamente o dobro de soluções do que a representação SD10, mas o desvio percentual entre ambas é muito semelhante.

Percebe-se que para os problemas nos quais o desvio percentual foi inferior a 1% os resultados foram idênticos, com exceção dos exemplares LA01 e LA02.

Outro ponto importante que deve ser notado, é a inclinação dos gráficos em cada representação. Percebe-se que em todos os casos, com a exceção dos exemplares LA01 e LA02, nos quais a curva para SD20 é crescente e SD10 é decrescente e os exemplares LA17, LA18, nos quais SD20 é um declive muito suave e SD10 é crescente, todas as outras inclinações são equivalentes.

Para uma visão rápida do comportamento do *makespan* das representações SD20 e SD10 é apresentado o gráfico gerado com os dados das Tabelas 16 e 17, visualizado na Figura 45.

Figura 44 – Valores mínimos de desvio obtidos para SD20 e SD10

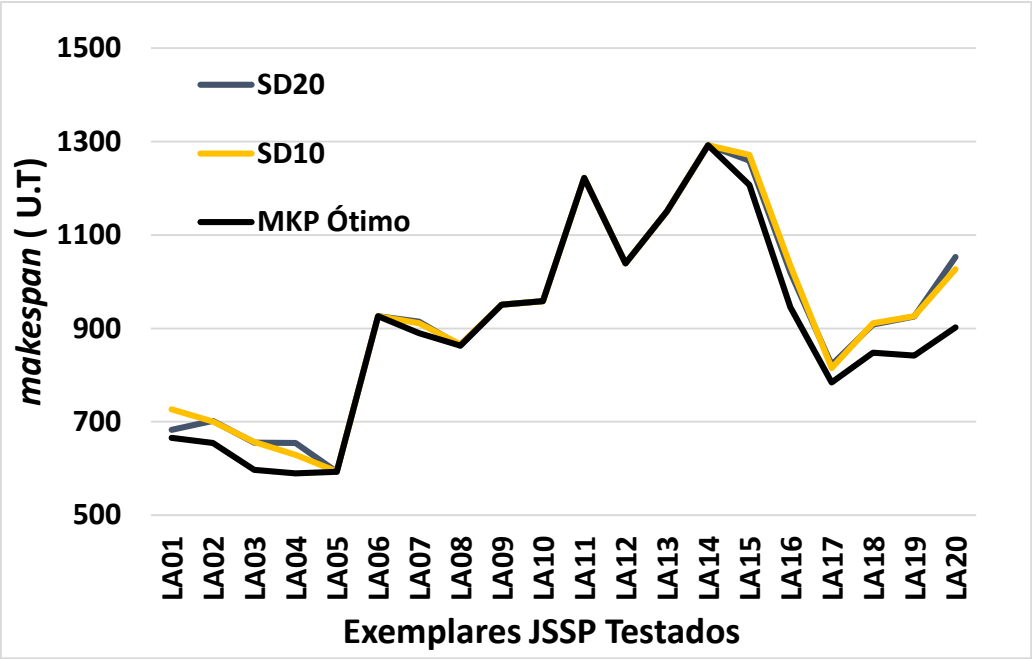


Fonte: O Autor

Nota-se, portanto, que não há uma relação significativa entre o aumento da quantidade de laços externos e a melhora no desempenho da representação, no que tange à redução no desvio percentual em relação ao ótimo.

Percebe-se também, devido às características da representação por SD, que manter uma quantidade de laços média (entre 10 e 20) é saudável para a convergência da solução. Em outras palavras não é necessário aumentar a quantidade de laços, além do que foi mostrado, caso isso fosse feito não se notaria grandes benefícios relativos a uma convergência mais rápida ou ainda a um *makespan* mais próximo ao MKP*, entretanto ocasionaria um aumento no custo computacional, pois com o aumento de laços externos com o objetivo de se refinar mais a solução ter-se-ia um aumento no número de soluções o que resultaria num tempo de processamento maior.

Figura 45 – *Makespan* para as representações SD20 e SD10



Fonte: O Autor

5.3 COMPARAÇÃO DAS REPRESENTAÇÕES LISTA, OB E SD10

Seguindo o mesmo raciocínio da seção anterior, as Tabelas 18 e 19 são visualizados os valores mínimos, máximos e médios encontrados em cada um dos problemas testados para as representações OB e LISTA, os valores em negrito indicam que o algoritmo de otimização encontrou o valor ótimo de *makespan* conhecido, quando os valores de máximo, mínimo e médio forem iguais, significa que nas dez rodadas os valores foram idênticos.

São apresentados na sequência os gráficos gerados a partir das Tabelas 18 e 19, nas Figuras 46 e 47, para uma visualização de como foi a distribuição das dez rodadas de cada experimento.

Tabela 18 – Representação OB – Valores de *makespan* obtidos

Problema	Mínimo	Média	Máximo
LA01	**666**	674,9	692
LA02	667	692,1	734
LA03	615	632,1	674
LA04	597	624,1	632
LA05	**593**	593	593
LA06	**926**	926	926
LA07	890	894,3	916
LA08	**863**	863	863
LA09	**951**	951	951
LA10	**958**	958	958
LA11	**1222**	1222	1222
LA12	**1039**	1039	1039
LA13	**1150**	1150	1150
LA14	**1292**	1292	1292
LA15	1207	1219,5	1255
LA16	987	1016,9	1035
LA17	793	811,7	848
LA18	861	896,6	932
LA19	876	898,7	911
LA20	914	929,1	958

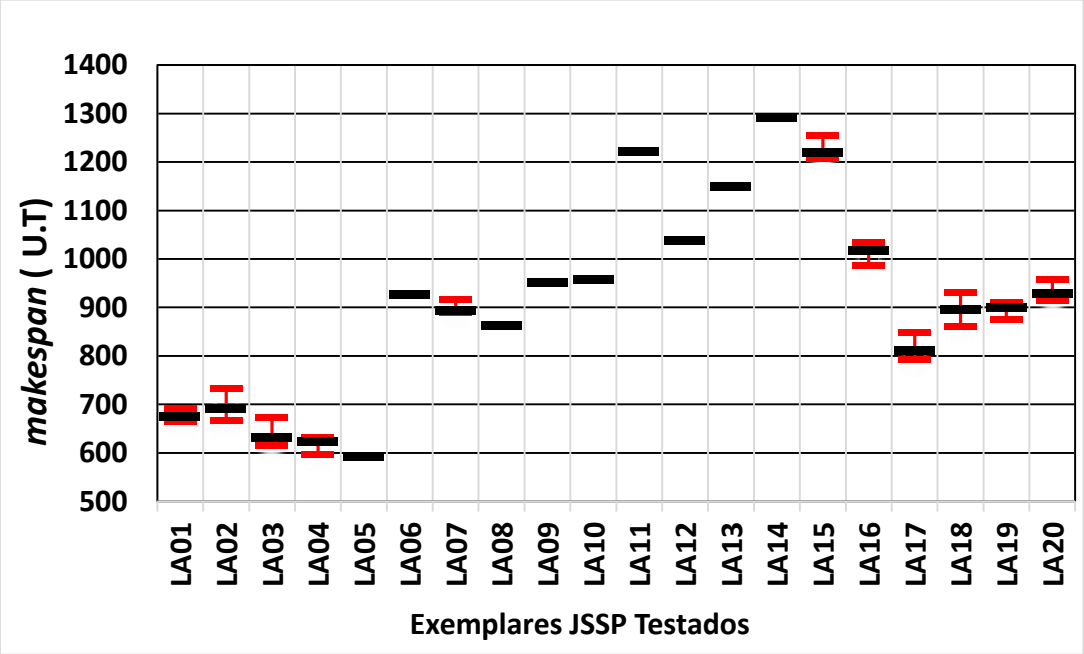
Fonte: O Autor

Tabela 19 – Representação LISTA – Valores de *makespan* obtidos

Problema	Mínimo	Média	Máximo
LA01	681	726,5	765
LA02	734	754,7	813
LA03	634	684,6	711
LA04	650	716,9	761
LA05	**593**	593,6	596
LA06	**926**	929,5	944
LA07	953	978,2	1014
LA08	**863**	900,5	940
LA09	**951**	956,4	960
LA10	**958**	958	958
LA11	**1222**	1222	1222
LA12	**1039**	1044	1060
LA13	**1150**	1150	1150
LA14	**1292**	1292	1292
LA15	1370	1404	1439
LA16	1051	1106,9	1135
LA17	848	866,8	900
LA18	932	951,7	988
LA19	970	996,1	1025
LA20	1160	1172,9	1199

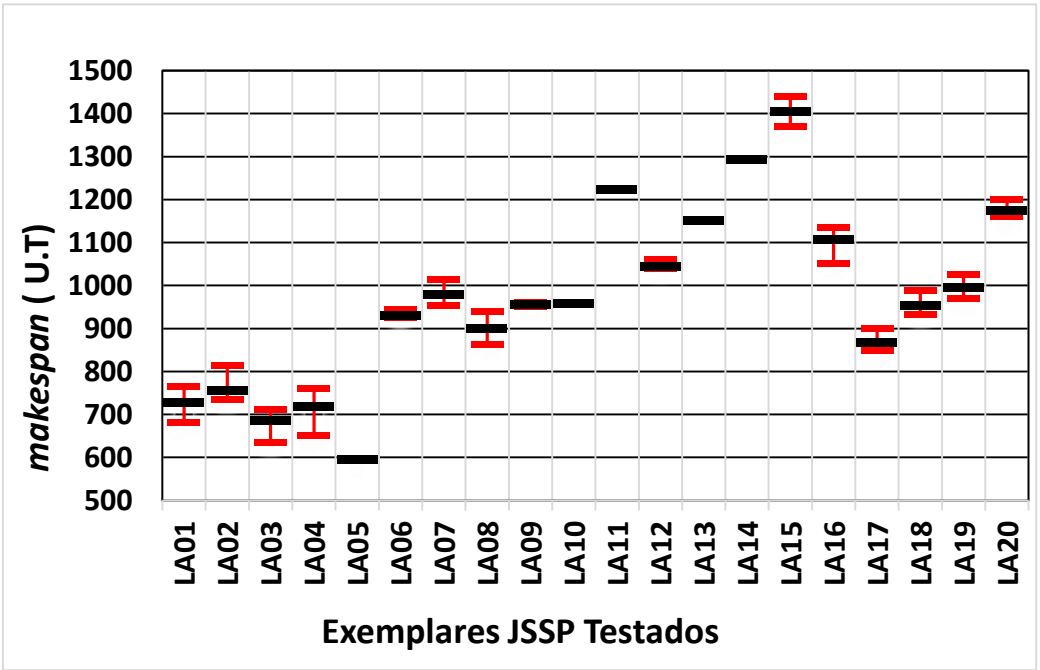
Fonte: O Autor

Figura 46 – Representação OB – Valores mínimos, médios e máximos do *makespan* obtidos



Fonte: O Autor

Figura 47 – Representação LISTA – Valores mínimos, médios e máximos do *makespan* obtidos



Fonte: O Autor

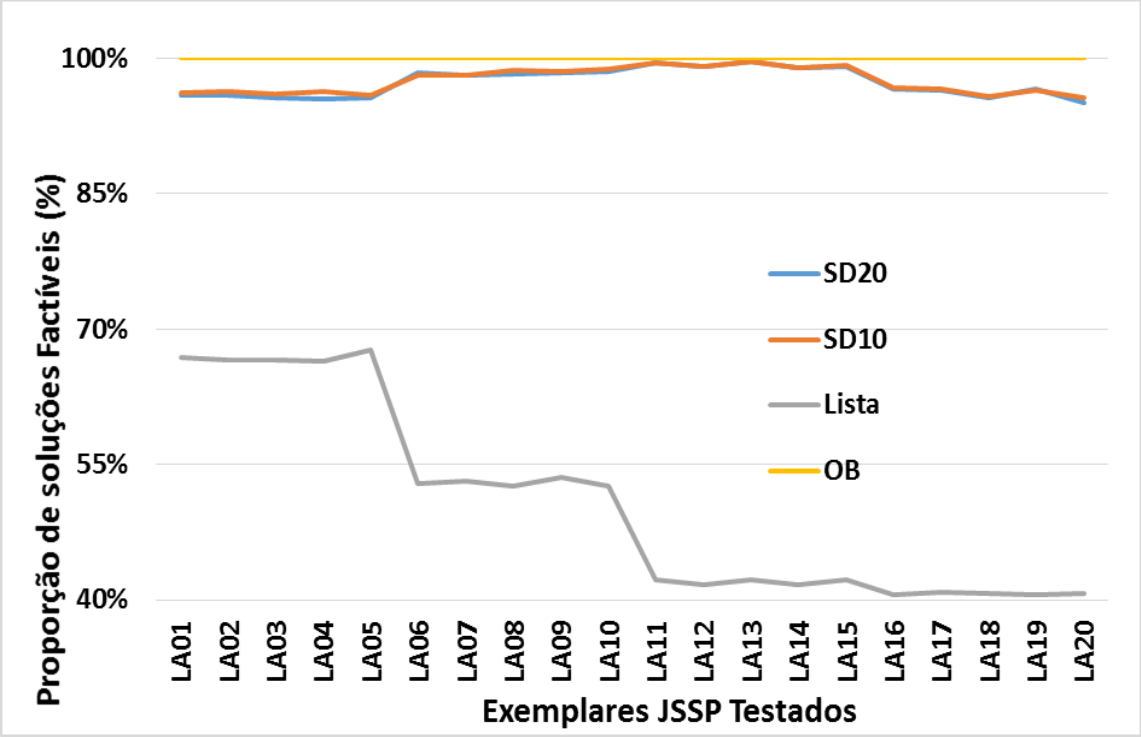
Conforme apresentado na seção 3.4.1, pg. 57, a representação OB adota uma interpretação do cromossomo que permite gerar apenas soluções factíveis. Por outro lado, a representação por LISTA gera um número muito grande de soluções não factíveis, e ambas são representações da solução de forma direta, por sua vez a representação por SD utiliza uma representação da solução de forma indireta, mas também se baseia em lista de preferências, e consegue-se atingir um número baixo de soluções não factíveis, como mostra o gráfico da Figura 48, baseado na Tabela 20.

Tabela 20 – Percentual de soluções por tipo de representação

Problema	SD 20 %	SD 10 %	LISTA %	OB %
LA01	95,92	96,17	66,8	100
LA02	95,92	96,3	66,58	100
LA03	95,62	95,98	66,55	100
LA04	95,43	96,31	66,47	100
LA05	95,64	95,93	67,75	100
LA06	98,37	98,17	52,83	100
LA07	98,12	98,07	53,21	100
LA08	98,29	98,73	52,66	100
LA09	98,39	98,46	53,64	100
LA10	98,49	98,87	52,55	100
LA11	99,49	99,52	42,27	100
LA12	99,14	99,15	41,64	100
LA13	99,59	99,57	42,24	100
LA14	98,97	98,99	41,68	100
LA15	99,09	99,26	42,17	100
LA16	96,66	96,77	40,5	100
LA17	96,47	96,58	40,86	100
LA18	95,56	95,73	40,65	100
LA19	96,63	96,5	40,59	100
LA20	95,12	95,68	40,71	100

Fonte: Autor

Figura 48 – Proporção de soluções factíveis por tipo de representação



Fonte: O Autor

Verificando ainda a Figura 48, nota-se que a representação por SD, gera uma quantidade superior aos 95% de soluções factíveis, em todos os exemplares LA testados, enquanto a representação por LISTA no máximo atinge os 67,75% de soluções factíveis até o exemplar LA05, decaindo após o LA05 em todos os exemplares e chegando ao LA20 com apenas 40% de soluções factíveis.

Outro ponto interessante, é quando se compara a quantidade total de soluções geradas (factíveis ou não). Constata-se que a representação por LISTA gera uma quantidade de soluções equivalente à representação OB. Justificando inclusive o tempo de processamento da representação por LISTA ser maior que das outras representações como pode ser visto nas Tabelas 21 e 22, devido ao fato de gerar uma quantidade muito superior de soluções não factíveis quando comparada, percentualmente, com as representações OB e SD.

Tabela 21 – Comparação entre quantidades de soluções, tempo de processamento (segundos) e tempo unitário (milissegundos)

	OB		
Problema	Total de Soluções	T.Proc	T. Unitário
LA01	18596,1	8,349	0,45
LA02	18645,7	9,037	0,48
LA03	18619,4	8,256	0,44
LA04	18634,1	7,836	0,42
LA05	18632,9	7,566	0,41
LA06	18992,3	12,51	0,66
LA07	18985,5	12,33	0,65
LA08	18985,5	12,01	0,63
LA09	18999,9	12,36	0,65
LA10	19023,8	12,51	0,66
LA11	19264,1	19,34	1,00
LA12	19255,7	16,73	0,87
LA13	19248,2	17,93	0,93
LA14	19246,3	20,41	1,06
LA15	19244	13,41	0,70
LA16	19250,7	12,99	0,67
LA17	19259,7	16,73	0,87
LA18	19252,9	17,68	0,92
LA19	19245,6	17,4	0,90
LA20	19251,3	14,14	0,73

Fonte: O Autor

Tabela 22 – Comparação entre quantidades de soluções, tempo de processamento (segundos) e tempo unitário (milissegundos)

	LISTA		
Problema	**Total de Soluções**	**T.Proc**	**T. Unitário**
LA01	18611,3	20,63	1,11
LA02	18606,2	35,27	1,90
LA03	18625,2	34,17	1,83
LA04	18642,6	35,90	1,93
LA05	18650,9	30,43	1,63
LA06	18986,5	52,77	2,78
LA07	19003,1	55,31	2,91
LA08	19002,8	85,62	4,51
LA09	18984,5	57,71	3,04
LA10	18986,8	60,47	3,19

Fonte: O Autor

Tabela 22 – Comparação entre quantidades de soluções, tempo de processamento (segundos) e tempo unitário (milissegundos) (cont.)

	LISTA		
Problema	**Total de Soluções**	**T.Proc**	**T. Unitário**
LA11	19249,2	173,81	9,03
LA12	19255	148,08	7,69
LA13	19259,7	169,52	8,80
LA14	19256,8	165,40	8,59
LA15	19263,5	168,92	8,77
LA16	19262,4	86,77	4,50
LA17	19269,4	68,60	3,56
LA18	19255,4	49,86	2,59
LA19	19251,4	48,27	2,51
LA20	19264,7	56,79	2,95

Fonte: O Autor

Na Figura 49 visualiza-se o gráfico das colunas do tempo unitário das duas representações OB e LISTA extraídos das Tabelas 21 e 22, mostrando a diferença de tempo de processamento para cada representação, bem como a ideia de custo computacional de ambas. Percebe-se, como já foi mencionado, que a quantidade de soluções geradas é equivalente, em torno de 18000 e 19000 para ambas as representações, porém a representação por LISTA gera um número excessivo de soluções não factíveis, como visto na Figura 49, o que pode influenciar também no tempo de processamento.

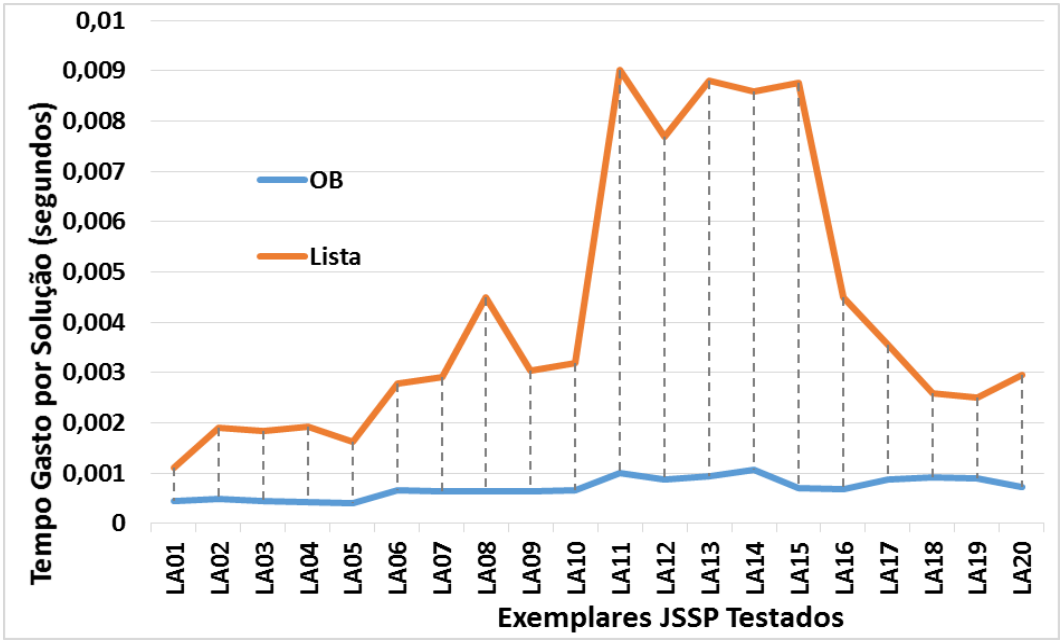
Tabela 23 – Proporção entre os tempos unitários

Problema	Proporção LISTA/OB	Problema	Proporção LISTA/OB
LA01	2,47	LA11	9,00
LA02	3,91	LA12	8,85
LA03	4,14	LA13	9,45
LA04	4,58	LA14	8,10
LA05	4,02	**LA15**	**12,58**
LA06	4,22	LA16	6,67
LA07	4,48	LA17	4,10
LA08	7,12	LA18	2,82
LA09	4,67	LA19	2,77
LA10	4,85	LA20	4,01

Fonte: Autor

Por exemplo para o problema LA15, constata-se um tempo de processamento 12,58 vezes maior, para a representação por LISTA, quando se compara com a representação OB, os dados das proporções de tempo entre as duas representações são mostrados na Tabela 23.

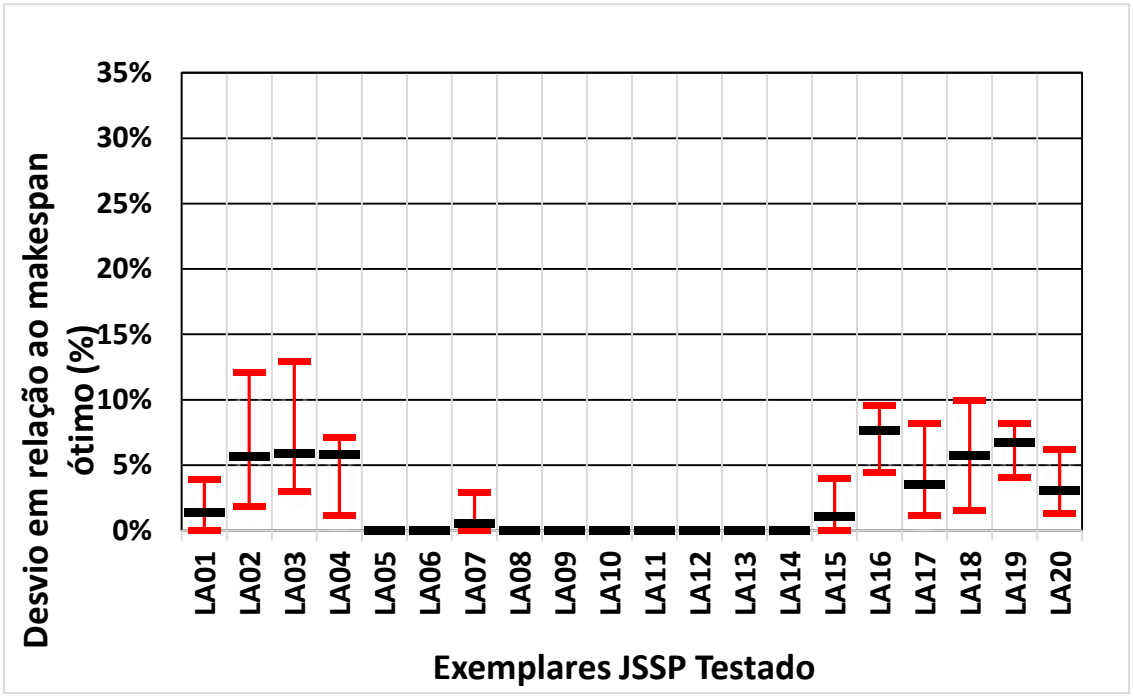
Figura 49 – Comparação do tempo unitário gasto pelas representações OB e LISTA



Fonte: Autor

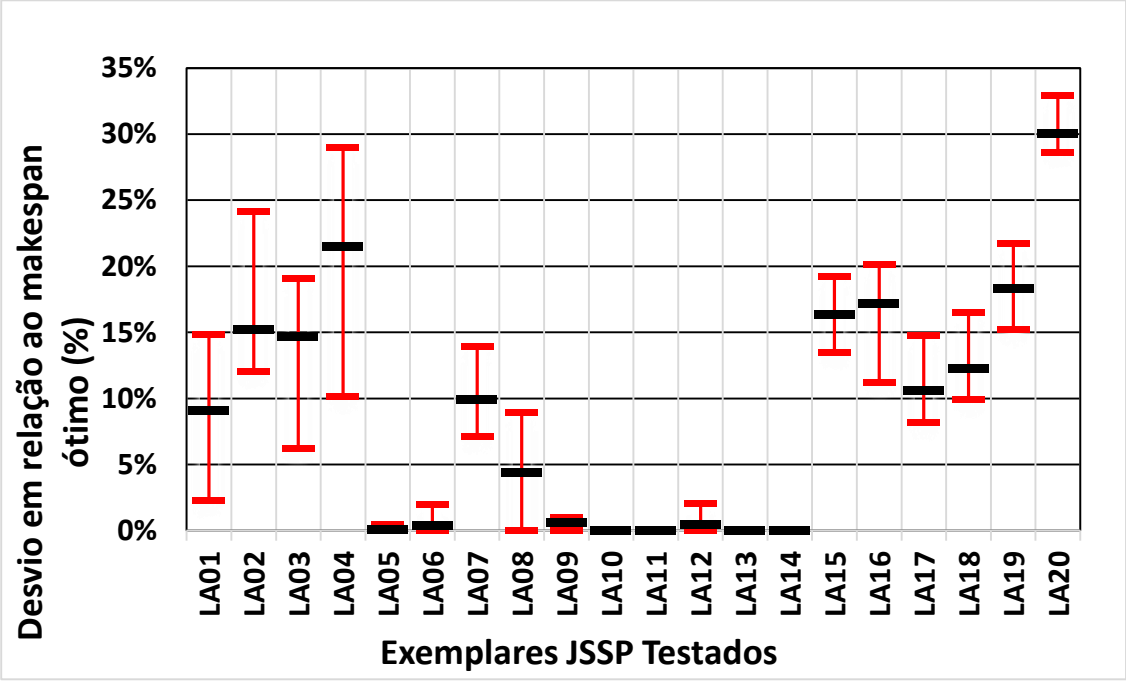
Os desvios percentuais da representação OB e da representação por LISTA em relação ao MKP*, são apresentados nas Figuras 50 e 51. Nota-se, novamente, desempenho melhor na representação OB, pois apresenta em todos os casos valores mínimos e médios menores que os da representação por LISTA. Observa-se ainda que a representação OB encontra o MKP* para exemplares LA01, e LA05 até LA15, enquanto que a representação por LISTA somente nos problemas LA05, LA06, LA08 até LA14, mais uma vez a representação OB tem desempenho melhor.

Figura 50 – Representação OB – Valores mínimos, médios e máximos de desvio obtidos



Fonte: O Autor

Figura 51 – Representação LISTA – Valores mínimos, médios e máximos de desvio obtidos



Fonte: O Autor

5.4 COMPARANDO AS REPRESENTAÇÕES OB E SD10

O total de soluções média para as dez simulações de cada exemplar das representações OB e SD10 são vistos nas Tabelas 24 e 25, bem como o tempo de processamento (T.Proc) e o tempo unitário (T.Unitário) que é uma relação entre o tempo de processamento e o total de soluções.

Tabela 24 – Comparação entre quantidades de soluções, tempo de processamento (segundos) e tempo unitário (milissegundos)

	OB		
Problema	Total de Soluções	T.Proc	T. Unitário
LA01	18596,1	8,349	0,45
LA02	18645,7	9,037	0,48
LA03	18619,4	8,256	0,44
LA04	18634,1	7,836	0,42
LA05	18632,9	7,566	0,41
LA06	18992,3	12,51	0,66
LA07	18985,5	12,33	0,65
LA08	18985,5	12,01	0,63
LA09	18999,9	12,36	0,65
LA10	19023,8	12,51	0,66
LA11	19264,1	19,34	1,00
LA12	19255,7	16,73	0,87
LA13	19248,2	17,93	0,93
LA14	19246,3	20,41	1,06
LA15	19244	13,41	0,70
LA16	19250,7	12,99	0,67
LA17	19259,7	16,73	0,87
LA18	19252,9	17,68	0,92
LA19	19245,6	17,4	0,90
LA20	19251,3	14,14	0,73

Fonte: O Autor

Na Figura 52, visualiza-se o gráfico baseado nas colunas do tempo unitário das duas representações, OB e SD10 extraídos das Tabelas 24 e 25 evidenciando a diferença de tempo de processamento para cada representação, bem como a ideia de custo computacional de ambas.

Quando se compara o desempenho das representações SD10 e a representação OB, relativo ao tempo unitário, percebe-se que os tempos de execução são muito próximos.

Tabela 25 – Comparação entre quantidades de soluções, tempo de processamento (segundos) e tempo unitário (milissegundos)

	SD10		
Problema	Total de Soluções	T.Proc	T. Unitário
LA01	21826,8	14,71	0,67
LA02	23683,4	15,86	0,67
LA03	20468	13,39	0,65
LA04	25254,4	16,37	0,65
LA05	19386,6	11,93	0,62
LA06	29786,8	23,46	0,79
LA07	37068,7	30,25	0,82
LA08	32782,7	24,00	0,73
LA09	31207,4	21,10	0,68
LA10	28626,7	15,20	0,53
LA11	39754	26,31	0,66
LA12	39404	24,13	0,61
LA13	38997	37,03	0,95
LA14	38508	42,74	1,11
LA15	60239	68,29	1,13
LA16	41799,7	47,37	1,13
LA17	46180,6	46,07	1,00
LA18	44879,5	50,99	1,14
LA19	43261,2	37,84	0,87
LA20	44506,4	37,07	0,83

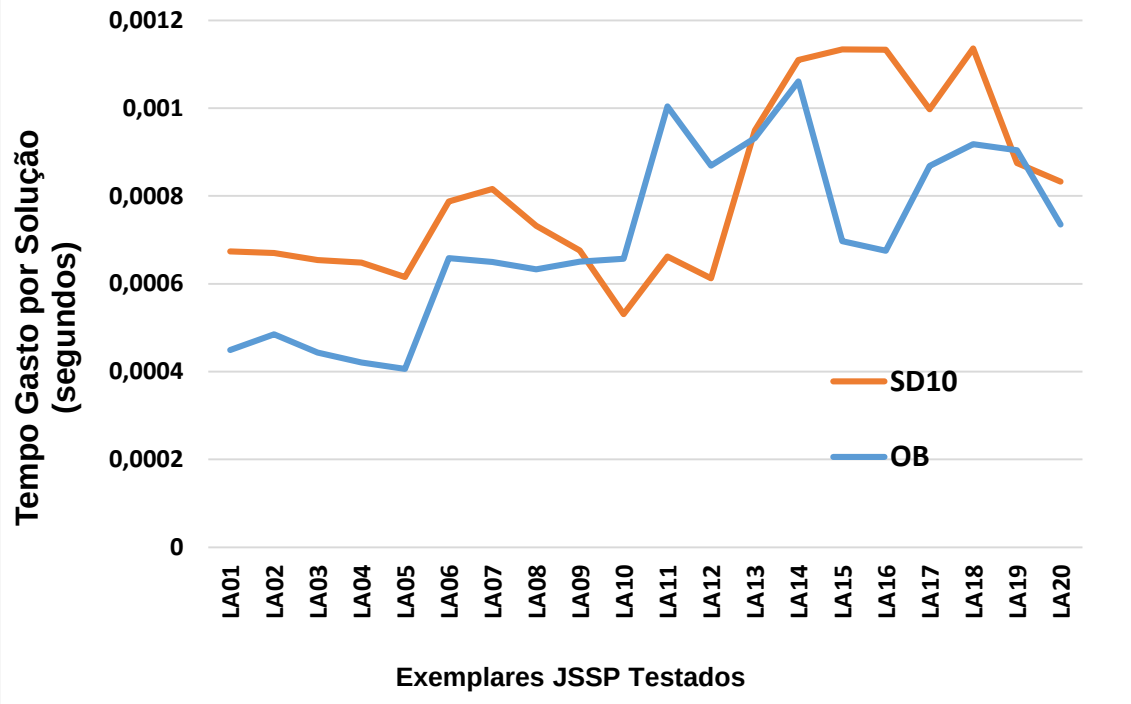
Fonte: O Autor

Como se percebe, a diferença do tempo de processamento quando se compara as representações OB e SD10, são bem menores, do que quando comparamos as representações OB com LISTA, conforme visualiza-se na Tabela 26. Para o exemplar LA16, por exemplo, o qual representa a maior diferença, o tempo do SD10 é 1,68 vezes maior.

Outra observação importante é que na representação SD10, o total de soluções geradas sempre é maior, isso se explica, pois depende do critério de

parada adotado. No caso LA15, por exemplo, esse número chega a ser mais que o triplo do número de soluções geradas na representação OB, entretanto o desempenho é semelhante.

Figura 52 – Comparação do tempo unitário gasto pelas representações OB e SD10



Fonte: O Autor

Outro ponto que deve ser salientado é que a robustez do método utilizado na SD, exige que seja fixado uma quantidade de laços externos e internos, influenciando no tempo de processamento total desta representação, sendo assim, verifica-se que o tempo de processamento também é proporcionalmente equivalente entre as duas representações, por exemplo o exemplar LA20, na representação OB são geradas em torno de 19251 soluções em 14,14 segundos, enquanto na SD10, são gerados em torno de 44505 em 37,06 segundos.

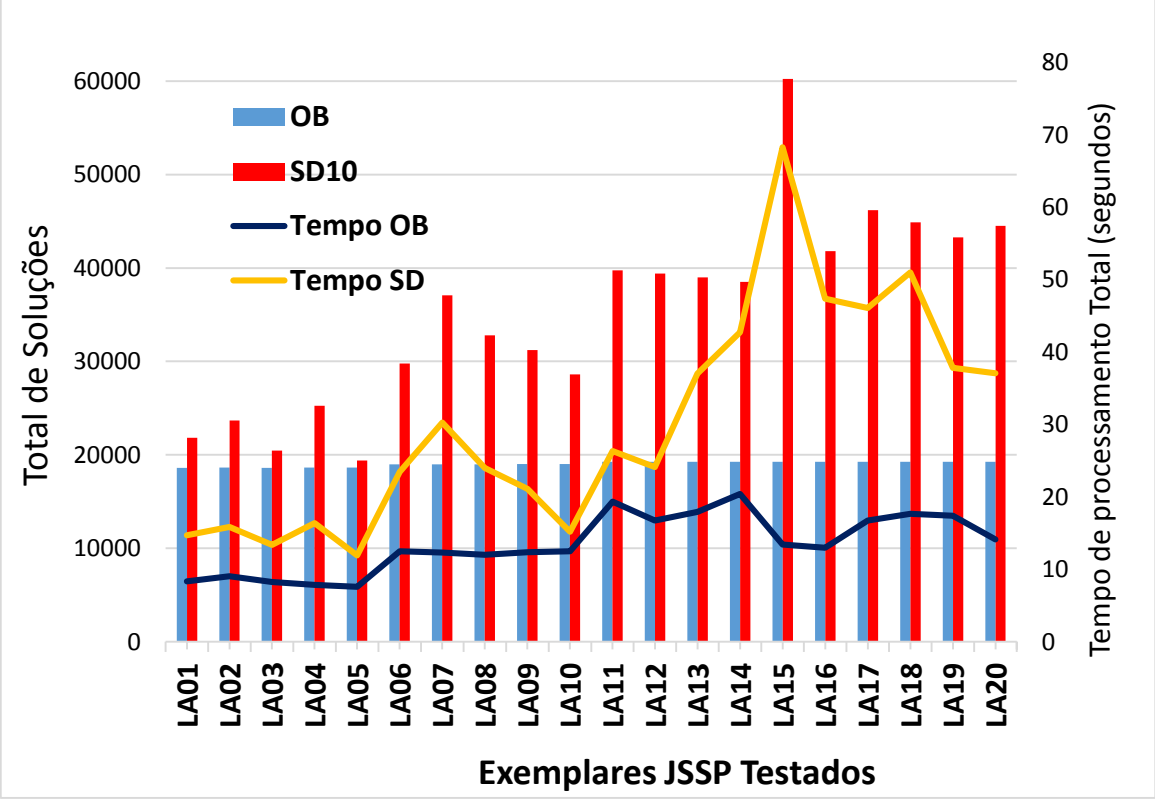
Tabela 26 – Proporção entre os tempos unitários

Problema	Proporção SD/OB	Problema	Proporção SD/OB
LA01	1,50	LA11	0,66
LA02	1,38	LA12	0,70
LA03	1,48	LA13	1,02
LA04	1,54	LA14	1,05
LA05	1,52	LA15	1,63
LA06	1,20	**LA16**	**1,68**
LA07	1,26	LA17	1,15
LA08	1,16	LA18	1,24
LA09	1,04	LA19	0,97
LA10	0,81	LA20	1,13

Fonte: Autor

A tendência entre as duas representações em relação ao tempo e o número de soluções é visualizada na Figura 53, extraída das Tabelas 24 e 25.

Figura 53 – Total de soluções e o tempo de processamento



Fonte: O Autor

Analisando a Figura 53, percebe-se que na representação OB a quantidade soluções praticamente se matem constante (barras azuis). Por outro lado, na representação SD conforme aumenta-se o número de soluções o tempo também aumenta proporcionalmente, lembrando é claro que o SD utiliza o número de laços externos como critério de parada e laços internos como critério de convergência, isso faz com que o método continue mesmo quando a solução já foi encontrada.

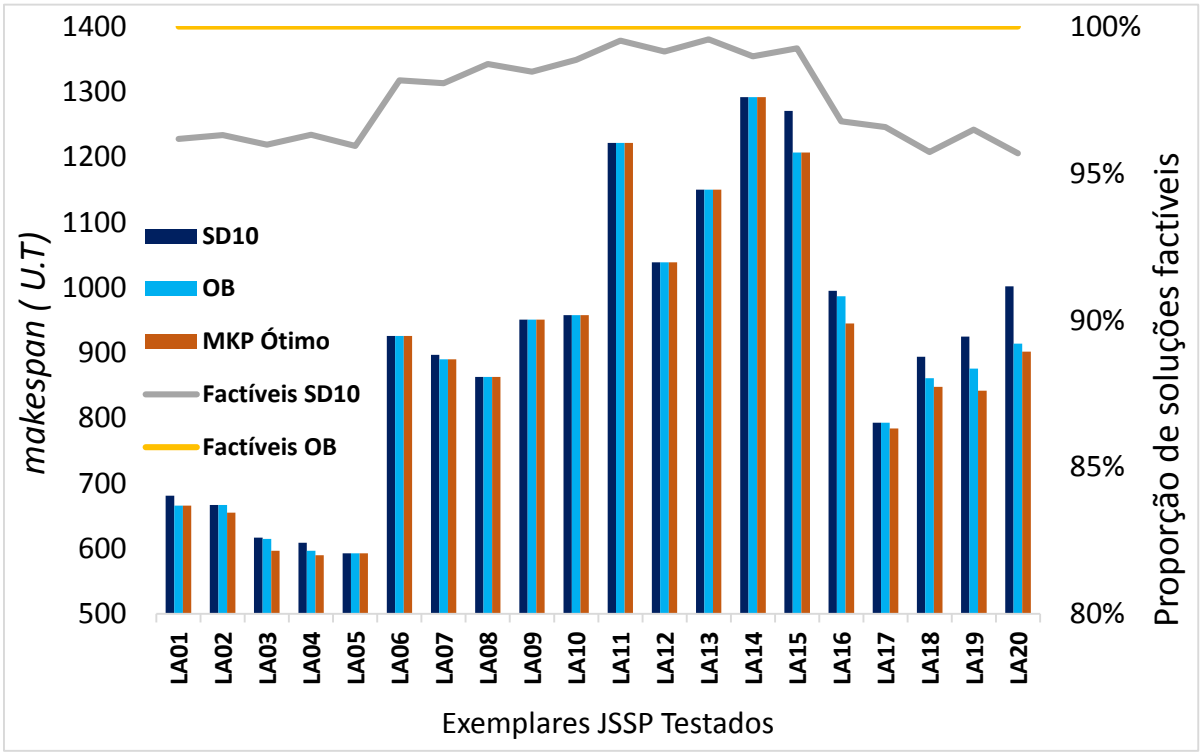
Neste ponto tende-se a concluir que representação OB é muito mais veloz, sendo assim, para a devida constatação é interessante investigar como se processou a convergência dos métodos em ambas as representações, o que será avaliado na próxima seção.

5.5 ANÁLISE DA CONVERGÊNCIA DOS MÉTODOS UTILIZADOS PARA OB E SD10

Na Figura 54, é visualizado o gráfico do melhor *makespan* alcançado das representações SD10, OB e também o MKP*, bem como a comparação do percentual de soluções factíveis de ambas as representações.

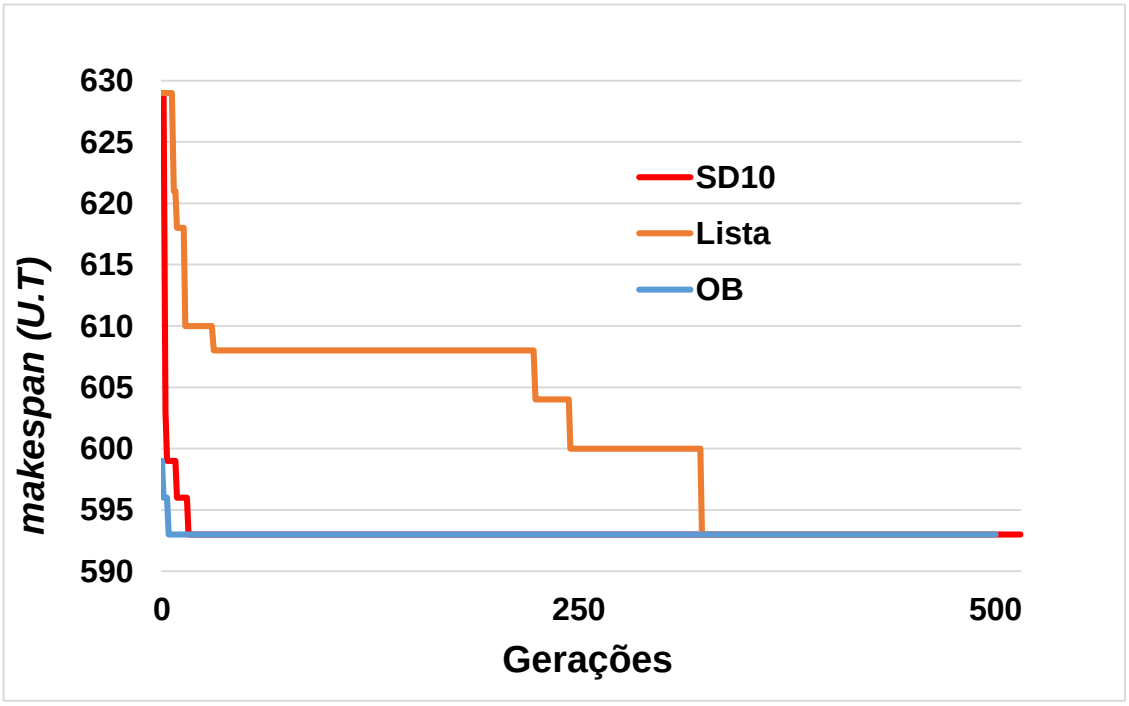
Para a análise da convergência do método adotado em cada representação, comparam-se amostras de exemplares das representações SD10, LISTA e OB. Escolheu-se para a análise dois estados, primeiro quando ambos métodos atingem o MKP*, e segundo quando nenhum dos métodos chega ao MKP*. Sendo assim para a devida analise foram escolhidos o exemplar LA05 e LA06 onde ambos os métodos chegaram no MKP* e os exemplares LA16 e LA17 onde nenhum dos métodos apresentados chegou no MKP*. Observa-se que para cada exemplar da família LA foram efetuadas 10 simulações, no entanto para se gerar o gráfico da convergência, escolheu-se a simulação que gerou o menor *makespan* no menor tempo. Os gráficos dos exemplares escolhidos são visualizados nas Figuras 55, 56,57 e 58.

Figura 54 – Comparação do *makespan* e quantidade de soluções factíveis



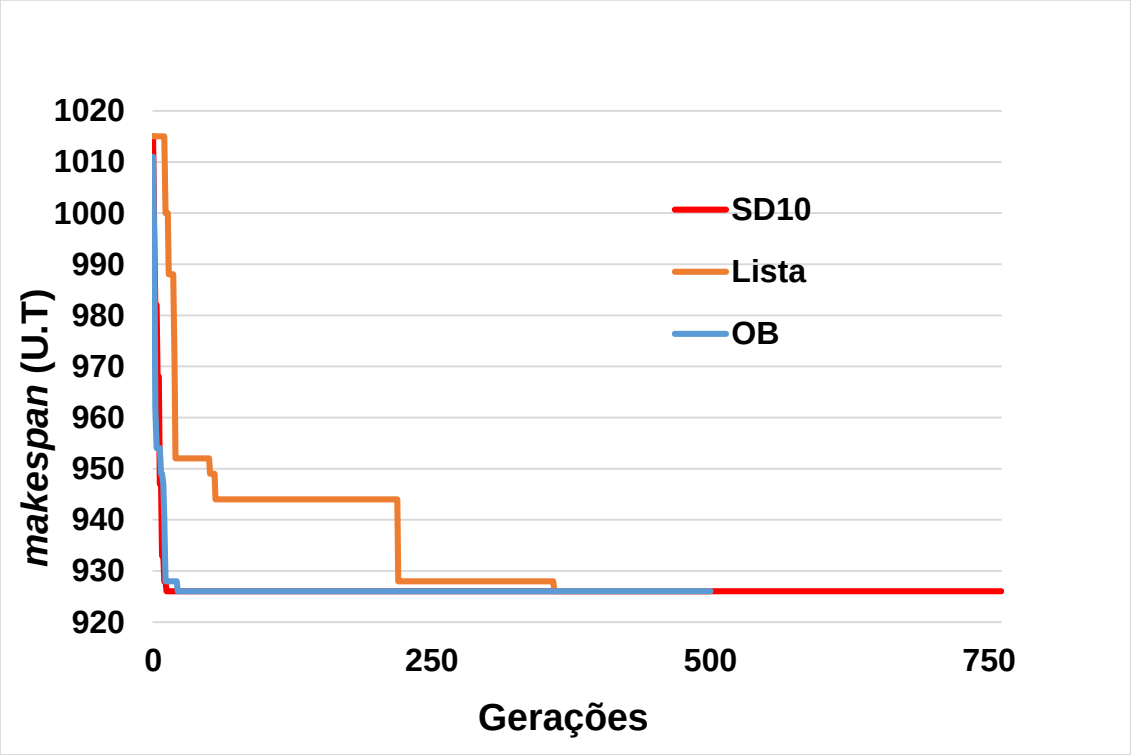
Fonte: O Autor

Figura 55 – Gráfico da convergência para o problema LA05



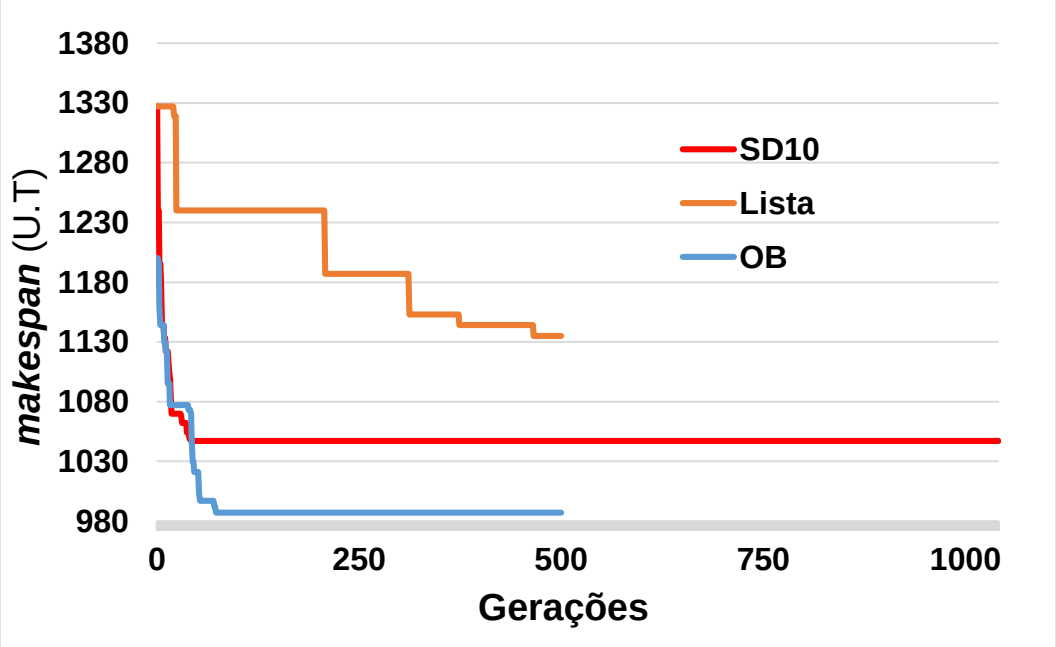
Fonte: O Autor

Figura 56 – Gráfico da convergência para o problema LA06



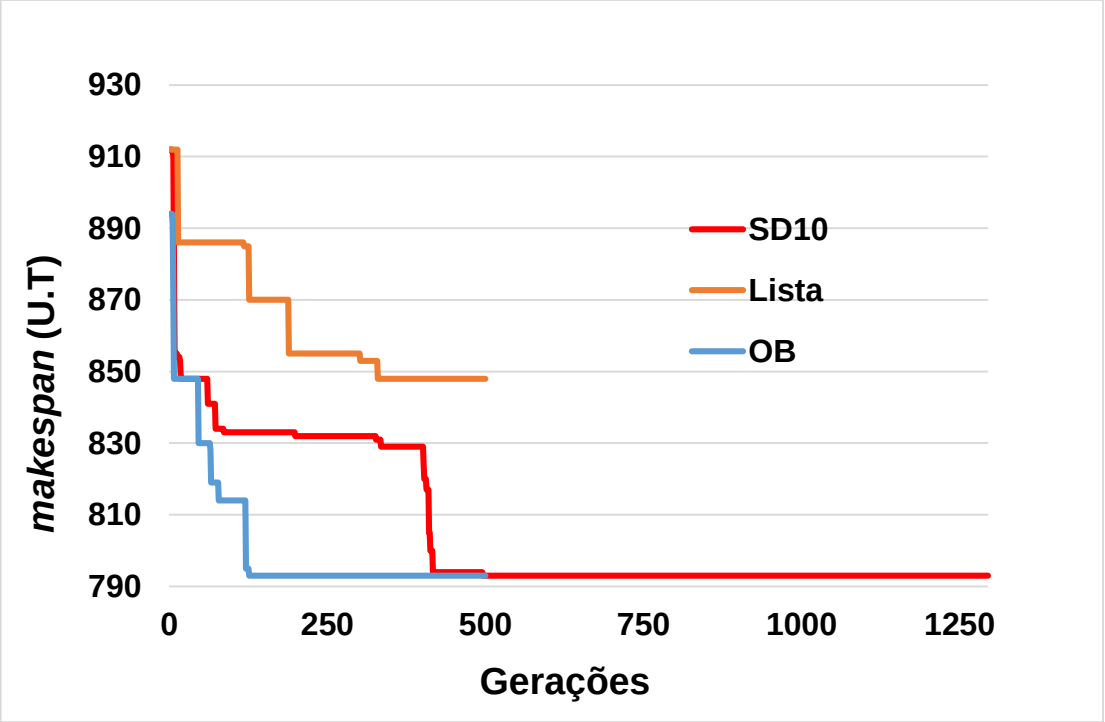
Fonte: O Autor

Figura 57 – Gráfico da convergência para o problema LA16



Fonte: O Autor

Figura 58 – Gráfico da convergência para o problema LA17



Fonte: O Autor

Para todos os exemplares escolhidos percebe-se que a convergência da representação OB dá-se mais rapidamente que para as demais representações avaliadas. Observa-se, que nos exemplares analisados a representação OB inicia o processo de busca com um *makespan* menor, devido as características da representação OB, levando assim uma certa vantagem inicialmente.

Observa-se ainda que para os valores em que ambas as representações atingirão o MKP*, a convergência é muito semelhante.

5.6 ANÁLISE DOS RESULTADOS

5.6.1 Influência dos operadores de cruzamento na representação SD

Os resultados obtidos mostram que, em geral, os operadores testados não apresentam grande influência nos resultados, o que foi observado utilizando os exemplares LA01 a LA05, que estão na mesma classe de 10 *job*s e 5 máquinas, conforme Tabela 6, pg. 89. Entretanto, foi possível observar que a taxa de cruzamento apresenta uma correlação positiva e próxima de 1,0, conforme apresentado no Quadro 8, pg. 92, com a proporção de soluções factíveis geradas no processo de busca favorecendo a convergência para valores de *makespan* mais adequados.

Portanto, pode-se concluir nesse caso que a taxa de cruzamento interfere na proporção de soluções factíveis geradas pela representação binária.

5.6.2 Desempenho da representação por LISTA

Após os vários experimentos, utilizando os exemplares LA01 a LA20 objetivando comparar o desempenho das representações aqui estudadas, verificou-se que a representação por LISTA, apresentou o pior desempenho, não sendo a mais apropriada para os exemplares *job shop* aqui apresentados, pois como se mostra na seção 5.3, pg. 110, a representação OB, se mostrou muito mais eficiente quando aplicada aos mesmos exemplares *job shop* e também favoreceu o processo de busca e a convergência para valores de *makespan* mais adequados.

5.6.3 Influência dos laços externos na representação SD

Conforme foi apresentado, efetuando experimentos com a representação SD10 e SD20, verificou-se desempenho equivalente. Não foi constatado que com o aumento de laços, a convergência para valores de *makespan* mais adequados foi influenciado, percebeu-se porém que com o aumento dos laços o tempo total da resolução até o melhor *makespan* achado tende a aumentar.

Conforme descrito na seção 5.2, pg. 102, existe um bom senso em mater o numero de laços com o valores entre 10 e 20, mostrando-se suficiente para uma convergência equivalente às demais representações.

5.6.4 Comparação do desempenho das representações OB e SD10

Constatou-se equivalência muito próxima entre ambas representações comparando-se o desempenho computacional.

Pensou-se que a representação OB por apresentar 100% de soluções factivéis a mesma convergeria mais rápido, ou chegaria no MKP* em todos os exemplares, o que não ocorreu, pois a mesma gera várias vezes combinações com a mesma solução, o que não ocorre na representação SD10, que apesar de não apresentar soluções com combinações idênticas, não gera 100% de soluções factivéis.

Consequentemente, considerando os resultados semelhantes obtidos pelas representações OB e SD, com pequena superioridade para a OB e o péssimo desempenho da representação por lista de prioridades, pode-se concluir que representações cromossômicas, permitem a geração de um maior número de soluções factíveis e que produzem resultados melhores, no que tange o comportamento de convergência do processo de busca e à qualidade das soluções.

O proximo capitulo apresenta as conclusões finais e futuros trabalhos para continuidade da pesquisa.

6 CONCLUSÕES, LIMITAÇÕES E SUGESTÕES

Nesta dissertação, foi realizado um estudo comparativo considerando duas representações com base em números inteiros e uma representação binária, analisando o efeito do operador de cruzamento no desempenho do algoritmo genético, quando aplicado aos problemas de sequenciamento da produção em *job shop*. Os experimentos foram realizados no contexto do algoritmo genético com semente dinâmica, o qual é baseado em uma representação binária indireta da solução, comparando-se com as representações OB e por LISTA.

Ao adotar a representação binária, até então incomum para esse tipo de problema, nota-se que a representação SD permite a utilização dos operadores convencionais, *One Point Crossove*r (OP), *Uniform Crossover* (UN), *Even Odd Crossover* (ED), que em tese, são mais eficientes em relação à geração de novas soluções viáveis pelo cruzamento de outras soluções também viáveis, percebeu-se pelos resultados obtidos que, em geral, os operadores de cruzamento testados não apresentam grande influência nos resultados. Entretanto, foi possível observar que a taxa de cruzamento apresenta uma correlação positiva, e próxima de 1,0, com a proporção de soluções factíveis geradas no processo de busca favorecendo a convergência para valores de *makespan* mais adequados.

Avaliando o desempenho entre as representações SD, OB e LISTA, em geral conclui-se o seguinte, que quanto maior a proporção de soluções factíveis melhora-se o resultado, o que faz com que a representação OB, por gerar 100% de soluções factíveis, tenha um desempenho melhor, seguido pela representação SD e por último com o pior desempenho a representação por LISTA.

Analisando a convergência da solução, tanto da representação OB como SD, verificou-se que há uma equivalência na busca, e também na velocidade da convergência, ficando evidente que os critérios de parada para a representação SD devem ser revistos, para que os tempos de parada sejam equivalentes a representação OB.

6.1 TRABALHOS FUTUROS

Buscar explorar melhor a relação entre taxa de cruzamento e soluções factíveis.

Adotar um modelo baseado nas representação OB e SD, que gere 100% de soluções factivéis, mas com a versatilidade da SD, que não gera soluções idênticas e efetuar novos experimentos.

Estender a pesquisa da metodologia proposta neste trabalho para outros conjuntos de *JSSP*, e também em ambiente real de produção de tal forma que uma generalização do método utilizado, seja devidamente investigado.

REFERÊNCIAS BIBLIOGRÁFICAS

AATHI, M.; R, R. A Comparison of Artificial Bee Colony algorithm and Genetic Algorithm to Minimize the Makespan for Job Shop Scheduling. **Procedia Engineering**, v. 97, p. 1745–1754, 2014.

ABDELMAGUID, T. F. Representations in Genetic Algorithm for the Job Shop Scheduling Problem: A Computational Study. **Journal of Software Engineering and Applications**, v. 03, n. 12, p. 1155–1162, 2010.

ADAMS, J.; BALAS, E.; ZAWACK, D. The shifting bottleneck procedure for job shop scheduling problem. **O RSA J. Comuting,** v. 3, p. 149–156, 1991.

AKGÜN, D.; ERDO, P. GPU accelerated training of image convolution filter weights using genetic algorithms. **Applied Soft Computing**, v. 30, n. 2015, p. 585–594, 2015.

ALEXANDRE, E.; CUADRA, L.; SALCEDO-SANZ, S.; PASTOR-SÁNCHEZ, A.; CASANOVA-MATEO, C. Hybridizing Extreme Learning Machines and Genetic Algorithms to select acoustic features in vehicle classification applications. **Neurocomputing**, v. 152, n. 2015, p. 58–68, 2015.

ALHAZOV, A.; LEPORATI, A.; MAURI, G.; PORRECA, A. E.; ZANDRON, C. Space complexity equivalence of P systems with active membranes and Turing machines. **Theoretical Computer Science**, v. 529, n. 2014, p. 69–81, 2013.

ALKHATIB, H.; DUVEAU, J. Electrical Power and Energy Systems Dynamic genetic algorithms for robust design of multimachine power system stabilizers. **International Journal of Electrical Power and Energy Systems**, v. 45, n. 1, p. 242–251, 2012.

ALVAREZ-VALDES, R.; FUERTES, A.; TAMARIT, J. M.; GIMÉNEZ, G.; RAMOS, R. **A heuristic to schedule flexible job-shop in a glass factory.** European Journal of Operational Research. **Anais**...2005.

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. **Pesquisa Operacional**. Rio de Janeiro: Elsevier Editora, 2007. 523p.

AZAD, A.; BULUÇ, A.; POTHEN, A. **A Parallel Tree Grafting Algorithm for Maximum Cardinality Matching in Bipartite Graphs** Proceedings of the IPDPS. **Anais**...2015.

BAETEN, J. C. M.; LUTTIK, B.; VAN TILBURG, P. Reactive turing machines. **Information and Computation**, v. 231, p. 143–166, 2013.

BARNES, J. W.; CHAMBERS, J. B. (1996). **Flexible job shop scheduling by tabu search.** Graduate program in operations research and industrial

engineering, The University of Texas at Austin 1996; Technical report series: ORP96-09.

BEAN, J. C. **Genetic Algorithms and Random Keys for Sequencing and Optimization INFORMS Journal on Computing**, 1994.

BERTSIMAS, D.; TSITSIKLIS, J. **Simulated Annealing Statistical Science**, 1993.

BO, W.; WODECKI, M. Computers & Industrial Engineering Solving permutational routing problems by population-based metaheuristics. v. 57, p. 269–276, 2009.

BOSWELL, T. Smart card security evaluation: Community solutions to intractable problems. **Information Security Technical Report**, v. 14, n. 2, p. 57–69, 2009.

CANYURT, O. E.; OZTURK, H. K. Application of genetic algorithm (GA) technique on demand estimation of fossil fuels in Turkey. **Energy Policy**, v. 36, n. 7, p. 2562–2569, 2008.

CASTRO, L. N. DE. **Natural Computing: A Brief Survey of Ideas and Applications**, BIC 2005: International Symposium on Bio-Inspired Computing Johor, MY, 9 th September 2005. Disponível em: http://pt.slideshare.net/lndecastro/2005-natural-computing-concepts-and-applications. Acessado em mar. 2015.

CÉSAR TREJO ZÚÑIGA, E.; LÓPEZ CRUZ, I. L.; GARCÍA, A. R. Parameter estimation for crop growth model using evolutionary and bio-inspired algorithms. **Applied Soft Computing**, v. 23, n. 2014, p. 474–482, 2014.

CHAMBERS, J. B. **Classical and flexible job shop scheduling by tabu search**. 1996. 232 f. Dissertação (Mestrado) – University of Texas, Austin, 1996.

CHENG, R.; GEN, M.; TSUJIMURA, Y. A Tutorial Survey of Job-Shop scheduling Problems using Genetic Algorithms – I Representation, **Computers ind. Engng** Vol. 30, n. 4, p. 983–997,1996.

COBHAM, A. **The Intrinsic Computational Difficulty of Functions**. Proceedings of the Third International Congress for Logic, Methodology and Philosophy of Science. North-Holland Pub. Co., 1965.

CROCE, F. D.; TADEI, R.; VOLTA, G. A genetic algorithm for the job shop problems. **Computers and Operations Research**, 22, p. 12–14, 1995.

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

DARWIN, C. **The Origin of Species.** New York. Mentor Books, 1953.

DAVIS, L. Job shop scheduling with genetic algorithm. **In Proc. Of the First Int. Conf. On Genetic Algorithms** (Edited by J. Grefenstette), p. 136–140. Lawrence Erlbaum Associates, Hillsdale, NJ , 1985.

DAVIS, L. **Handbook on Genetic Algorithms**. Van Nostrand Reinhold. NY, 1991.

DE ANDRADE, E. L. **Introdução à pesquisa operacional: métodos e modelos para a análise de decisões**. LTC, 2009.

DEVLIN, K. **Os problemas do milênio**. 2. ed. Rio de Janeiro: Record, 2008.

DONG, N.; GE, D.; FISCHER, M.; HADDAD, Z. **A genetic algorithm-based method for look-ahead scheduling in the finishing phase of construction projects.** Advanced Engineering Informatics. **Anais**...Elsevier Ltd, 2012. Disponível em: <http://dx.doi.org/10.1016/j.aei.2012.03.004>

DONIS-DÍAZ, C. A.; MURO, A. G.; BELLO-PÉREZ, R.; MORALES, E. V. A hybrid model of genetic algorithm with local search to discover linguistic data summaries from creep data. **Expert Systems with Applications**, v. 41, n. 4, p. 2035–2042, 2014.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. The Ant System : Optimization by a colony of cooperating agents. v. 26, n. 1, p. 1–13, 1996.

DORNDORF, U.; PESCH, E. Evolution based learning in a job shop scheduling environment. **Computers & Operations Research**, v. 22, n. I, p. 25–40, 1995.

EDMONDS, J. Paths, trees, and flowers. **Canadian Journal of Mathematics**, v. 17, n. 3, p. 449–467, 1965.

EISELT, H. A.; SANDBLOM, C.-L. **Decision Analysis, Location Models, and Scheduling Problems**. 1. ed. Berlin: Springer, 2004.

ERICKSON, J.; SIAU, K. **Theoretical and practical complexity of modeling methodsCommunications of the ACM**, 2007.

FAGHIHI, V.; REINSCHMIDT, K. F.; KANG, J. H. Construction scheduling using Genetic Algorithm based on Building Information Model. **Expert Systems with Applications**, v. 41, n. 16, p. 7565–7578, 2014.

FAN, K.; ZHANG, R. An analysis of research in job shop scheduling problem (2000-2009). **IEEE International Conference on Advanced Management Science**, Chengdu (China), 1, p. 282–288, 2010.

FANG, H.; ROSS P.; CORNE, D. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. **In Proc. of the Fifth Int. Conf. on Genetic Algorithms**, p. 375–382. Morgan Kaufmann Publishers, San Mateo, Calif. 1993.

FANG, Q.; LI, B.; SUN, X; ZHANG, J.; ZHANG, JI. Computing the Least-core and Nucleolus for Threshold Cardinality Matching Games. In: **Web and Internet Economics**. Springer, 2014. p. 474–479.

FESTAL, P.; RESENDE, M. G. C. GRASP: An annotated bibliography. **Operations Research/ Computer Science Interfaces Series**, v. 15, p. 325–367, 2002.

FLEURY, A. Planejamento do projeto de pesquisa e definição do modelo teórico. In: MIGUEL, P. A. C. (Coord.). **Metodologia de pesquisa em engenharia de produção e gestão de operações**. 2. ed. São Paulo: Elsevier, 2012.

FOGEL, D. B. Na introduction to Evolutionary Computation, **Australian Journal of Intelligent Information Processing,** v. 1, n. 2 p. 34–42, 1994.

FORSYTH, P.; WREN, A. Ants can colour graphs. **Journal of the Operational Research Society**, 1997, v. 48, p. 295–305.

GAO, W.-S.; SHAO, C. Iterative Dynamic Diversity Evolutionary Algorithm for Constrained Optimization. **Acta Automatica Sinica**, v. 40, n. 11, p. 2469–2479, 2014.

GAREY, M. R.; JOHNSON, D.S.; SETHI, R. **The complexity of Flow shop and Job-Shop scheduling, Mathematics of Operations Research**, v. 1, n. 2, 1976, p. 117–129.

GEN, M.; TSUJIMURA, Y.; KUBOTA, E. Solving job-shop scheduling problem using genetic algorithms. **In Proc. of the 16th Int. Conf. on Computer and Industrial Engineering**, p. 576–579. Ashikaga, Japan, 1994.

GHOLAMI, R.; SHAHABI, M.; HAGHIFAM, M. An efficient optimal capacitor allocation in DG embedded distribution networks with islanding operation capability of micro-grid using a new genetic based algorithm. **International Journal of Electrical Power and Energy Systems**, v. 71, n. 2015, p. 335–343, 2015.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2008.

GOLDBERG, D. E. **Genetic algorithms in search, optimization and machine learning.** New York: Addison-Wesley, 1989.

GONÇALVES, J. F.; RESENDE, M. G. C. A biased random key genetic algorithm for 2D and 3D bin packing problems. **International Journal of Production Economics**, v. 145, n. 2, p. 500–510, 2013.

GONÇALVES, J. F.; RESENDE, M. G. C. A biased random-key genetic algorithm for the unequal area facility layout problem. **European Journal of Operational Research**, p. 1–22, 2015.

GONG, Y.-J. ; CHEN, W. -N.; ZHAN, Z. -H.; ZHANG, J.; LI, Y.; ZANG, Q.; LI, J. -J. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. **Applied Soft Computing**, v. 34, n. 1, p. 286–300, 2015.

GRASSI, F. **Otimização por Algoritmos Genéticos do sequenciamento de ordens de produção em ambientes Job Shop.** 2014. 127 f. Dissertação (Mestrado em Engenharia de Produção) - Universidade Nove de Julho, São Paulo, 2014.

GRASSI, F.; TRIGUIS, P. H.; PEREIRA, F. H. Dynamic Seed Genetic Algorithm to Solve Job Shop Scheduling Problems, **International Journal of Production Research**, 2014.

HALAVA, V. **Decidable and Undecidable Problems in Matrix Theory Vesa Halava**. 1. ed. Turku: TUCS Technical Report, 1997.

HAX, A.; CANDEA, D. **Production and Inventory Management**, Englewood Cliffs, N.J., Prentice-Hall, 1984.

HEINONEN, J.; PETTERSSON, F. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. **Applied Mathematics and Computation**, v. 187, n. 2, p. 989–998, 2007.

HENDERSON, D.; JACOBSON, S. H.; JOHNSON, A. W. **THE THEORY AND PRACTICE OF SIMULATED ANNEALING**. Boston: Kluwer Academic Publishers, 2003. v. 57.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operacional**. 9. ed. AMGH, 2013.

HOLLAND, J.H. **Adaptation in Natural and Artificial Systems**. 2. Ed. The MIT Press, 1992.

HOLSAPPLE, C. W.; JACOB, V. S.; PAKATH, R.; ZAVERI, J. S. Genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. **IEEE Transactions on Systems, Man and Cybernetics**, v. 23, n. 4, p. 953–972, 1993.

HOSEINI, P.; SHAYESTEH, M. G. Efficient contrast enhancement of images using hybrid ant colony optimisation , genetic algorithm , and simulated annealing. **Digital Signal Processing**, v. 23, n. 3, p. 879–893, 2012.

INGBER, L. Simulated annealing: Practice versus theory. **Mathematical and Computer Modelling**, v. 18, n. 11, p. 29–57, dez. 1993.

IVERS, B. **Job shop optimization through multiple independent particle swarms.** 2006. 110 f. Dissertação (Mestrado em Engenharia Elétrica) – Oklahoma State University, Stillwater, 2006.

JABBAR, M. A.; DEEKSHATULU, B. L.; CHANDRA, P. Classification of Heart Disease Using K- Nearest Neighbor and Genetic Algorithm. **Procedia Technology**, v. 10, p. 85–94, 2013.

JACKSON, D. E.; BICAK, M.; HOLCOMBE, M. A paradigm for self-organisation: New inspiration from ant foraging trails. **Romanian Journal of Information Science and Technology**, v. 11, p. 253–265, 2008.

JEZ, A.; OKHOTIN, A. Computational completeness of equations over sets of natural numbers. **Information and Computation**, v. 237, p. 56–94, 2014.

JIAN-MING, Z. Computability vs. Nondeterministic and P vs. NP. **arXiv preprint arXiv:1305.4029**, n. May, p. 1–24, 2013.

JIN, I. H.; LIANG, F. Use of SAMC for Bayesian analysis of statistical models with intractable normalizing constants. **Computational Statistics and Data Analysis**, v. 71, n. 2014, p. 402–416, 2014.

JIN, J.; JEONG, J. Optimization of a free-form building shape to minimize external thermal load using genetic algorithm. **Energy & Buildings**, v. 85, n. 2014, p. 473–482, 2014.

KARABOGA, D. An idea based on Honey Bee Swarm for Numerical Optimization. **Technical Report TR06, Erciyes University**, n. TR06, p. 10, 2005.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. **Proceedings of ICNN'95 - International Conference on Neural Networks**, v. 4, 1995a.

KENNEDY, J.; EBERHART, R. **Particle swarm optimization** Neural Networks, 1995. Proceedings., IEEE International Conference on. **Anais**...1995b.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science (New York, N.Y.)**, v. 220, p. 671–680, 1983.

KLUG, W. S.; CUMMINGS, M. R.; SPENCER, C. A. **CONCEPTOS DE GENÉTICA**. Octava ed.Madrid: Pearson, 2006.

KOULAMAS, C.; ANTONY, S.; JAEN, R. A survey of simulated annealing applications to operations research problems. **Omega**, v. 22, n. 1, p. 41–56, jan. 1994.

KUBOTA, A. Study on optimal scheduling for manufacturing system by genetic algorithms. **Master's thesis**, Ashikaga Institute of Technology, Ashikaga, Japan March 1995.

KULKARNI, R. V.; VENAYAGAMOORTHY, G. K. Particle swarm optimization in wireless-sensor networks: A brief survey. **IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews**, v. 41, p. 262–267, 2011.

KUNNATHUR, A. S.; SUNDARARAGHAVAN, P. S.; SAMPATH, S. Dynamic rescheduling using a simulation-based expert system. **Journal of Manufacturing Technology Management**, v. 15, n. 2, p. 199–212, 2004.

LAKSHMIPATHY, N.; WINKLMANN, K. “Global” graph problems tend to be intractable. **Journal of Computer and System Sciences**, v. 32, n. 3, p. 407–428, 1986.

LALLA-RUIZ, E.; GONZÁLEZ-VELARDE, J. L.; MELIÁN-BATISTA, B.; MORENO-VEJA, J. M. Biased random key genetic algorithm for the Tactical Berth Allocation Problem. **Applied Soft Computing Journal**, v. 22, p. 60–76, 2014.

LAWRENCE, S. **Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques** (Supplement). 1984. Tese (Doutorado em Administração Industrial) - Carnegie-Mellon University, Pittsburgh, 1984.

LI, J. Q.; PAN, Q. K. **Chemical-reaction optimization for solving fuzzy job-shop scheduling problem with flexible maintenance activities.** International Journal of Production Economics. **Anais**...Elsevier, 2013. Disponível em: <http://dx.doi.org/10.1016/j.ijpe.2012.11.005>

LI, J. Q.; PAN, Q. K.; TASGETIREN, M. F. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. **Applied Mathematical Modelling**, v. 38, n. 3, p. 1111–1132, 2014.

LIN, Y. K.; PFUND, M. E.; FOWLER, J. W. Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. **Computers & Operations Research**, v.38, p. 901–916, 2011.

LINDEN, R. **Algoritmos genéticos**. Rio de Janeiro: Ciência Moderna, 2012.

LIU, M.; LIU, Y.; HU, H.; NIE, L. Genetic algorithm and mathematical morphology based binarization method for strip steel defect image with non-uniform illumination. **Journal of Visual Communication and Image Representation**, 2015.

LÓPEZ, O. C.; BARCIA, R. M.; EYADA, O. Problema de programação da produção: um esquema de classificação. **Revista Produção**, v.5, p. 145–168, 1995.

LUKASZEWICZ, P. P. **Metaheuristics for job shop scheduling problem, comparison of effective methods**. 2005. 123f. Dissertação (Mestrado em Logística) - Aarhus School of Business, Aarhus, 2005.

MAN, K.; TANG, K.; KWONG, S. Genetic algorithms: concepts and applications. **IEEE Transactions on …**, v. 43, n. 5, 1996.

MANTZARIS, D.; ANASTASSOPOULOS, G.; ADAMOPOULOS, A. Genetic algorithm pruning of probabilistic neural networks in medical disease estimation. **Neural Networks**, v. 24, n. 8, p. 831–835, 2011.

MARCONI, M. A.; LAKATOS, E. M. **Fundamentos de metodologia científica**. 7. ed. São Paulo: Atlas, 2010.

MARTINS, R.A.. **Abordagens quantitativa e qualitativa.** In: CAUCHICK MIGUEL, P.A.C. (Coord.). **Metodologia de pesquisa em Engenharia de Produção e Gestão de Operações.** 2. ed. São Paulo: Elsevier, 2012.

METROPOLIS, N.; ROSENBLUTH, A.W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equation of State Calculations by Fast Computing Machines. **The Journal of Chemical Physics**, v. 21, n. 6, p. 1087–1092, 1953.

MICHALEWICZ, Z.; SCHOENAUER, M. **Evolutionary Algorithms for Constrained Parameter Optimization Problems Evolutionary Computation**, 1996.

MITCHELL, M. Genetic algorithms: An overview. **Complexity**, v. 1, p. 31–39, 1995.

MITCHELL, M. **An introduction to genetic algorithms**, 1998.

MITCHELL, T. M. **Machine Learning**. McGraw-Hill, 1997.

MODOLO, V.; MENEZES, F. M.; VAZ, F. K.; PAULA, D. O.; PEREIRA, F. H. Estudo da influência do operador genético de cruzamento no algoritmo genético binário aplicado a problemas de sequenciamento em job shop. **XXXV Iberian Latin América Congress on Computational Methods in Engineering - CILAMCE 2014**, v. 1, p. 1–15, 2015.

MOKOTOFF, E. Parallel machine scheduling problems: a survey. **Asia-Pacific Journal of Operational Research**, v.18, p. 193–242, 2001.

MORABITO NETO, R; PUREZA, V. Modelagem e simulação. In: MIGUEL, P. A. C. (Coord.). **Metodologia de pesquisa em engenharia de produção e gestão de operações**. 2. ed. São Paulo: Elsevier, 2012.

MUSSI, L.; NASHED, Y. S. G.; CAGNONI, S. GPU-based asynchronous particle swarm optimization. **Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11**, p. 1555, 2011.

NAKANO, R.; YAMADA, T. Conventional genetic algorithm for job shop problems. In 4th ICGA, p. 474 – 479,1991.

PACHECO, R. F.; SANTORO, M. C. **Proposta de classificação hierarquizada dos modelos de solução para o problema de job shop scheduling Genetics and Molecular Biology**, 1999.

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

PARK, Y.; CHUN, S.; KIM, B. Cost-sensitive case-based reasoning using a genetic algorithm: application to medical diagnosis. **Artificial intelligence in medicine**, v. 51, n. 2, p. 133–145, 2011.

PASCUAL, G. G.; LOPEZ-HERREJON, R. E.; PINTO, M.; FUENTES, L.; EGYED, A. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. **Journal of Systems and Software**, v. 103, n. 1, p. 392–411, 2015.

PFUND, M.; FOWLER, J.W.; GUPTA, J. N. D. A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. **Journal of the Chinese Institute of Industrial Engineers**, v.21, n. 3, p. 230-241, 2004.

PILTAN, M.; SHIRI, H.; GHADERI, S. F. Energy demand forecasting in Iranian metal industry using linear and nonlinear models based on evolutionary algorithms. **Energy Conversion and Management**, v. 58, n. 2012, p. 1–9, 2012.

PINEDO, M. L. **Planning and Scheduling in Manufacturing and Services**. New York, Springer New York, p. 537, 2009.

PINEDO, M. L. **Scheduling: Theory, Algorithms, and Systems**, New York, Springer New York, p. 673, 2012.

POLI, R.; KENNEDY, J.; BLACKWELL, T. **Particle swarm optimization Swarm Intelligence**, 2007.

POLI, R. An analysis of publications on particle swarm optimization applications. **Journal of Artificial Evolution and Applications**, 2008.

POURZEYNALI, S.; SALIMI, S.; KALESAR, H. E. Sharif University of Technology Robust multi-objective optimization design of TMD control device to reduce tall building responses against earthquake excitations using genetic algorithms. **Scientia Iranica**, v. 20, n. 2, p. 207–221, 2013.

PRABHAKAR, B.; DEKTAR, K. N.; GORDON, D. M. The Regulation of Ant Colony Foraging Activity without Spatial Information. **PLoS Computational Biology**, v. 8, 2012.

QING-DAO-ER-JI, R.; WANG, Y. A new hybrid genetic algorithm for job shop scheduling problem. **Computers & Operations Research**, v. 39, n. 10, p. 2291–2299, out. 2012.

QUINIOU, M. LE; MANDEL, P.; MONIER, L. Optimization of Drinking Water and Sewer Hydraulic Management : Coupling of a Genetic Algorithm and Two Network Hydraulic Tools. **Procedia Engineering**, v. 89, p. 710–718, 2014.

RECHARD, J.; BIGNON, A.; BERRUET, P.; MORINEAU, T. Verification and validation of a Work Domain Analysis with turing machine task analysis. **Applied Ergonomics**, v. 47, n. 2015, p. 265–273, 2015.

REEVES, C. R.; ROWE, J. E. **Genetic algorithms: principles and perspectives: a guide to GA theory**. Springer Science & Business Media, 2003. v. 20.

RODAMMER, F.A.; WHITE, JR. K.P. A Recent Survey Of Production Scheduling. **IEEE Transaction on Systems, Man and Cybernetics**. v. 18, n. 6, p. 841–851, 1998.

RUIZ, E.; ALBAREDA-SAMBOLA, M.; FERNÁNDEZ, E.; RESENDE, M. G. C. A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. **Computers & Operations Research**, v. 57, p. 95–108, 2015.

SANTHANAM, T.; PADMAVATHI, M. S. Application of K-Means and Genetic Algorithms for Dimension Reduction by Integrating SVM for Diabetes Diagnosis. **Procedia - Procedia Computer Science**, v. 47, p. 76–83, 2015.

SELS, V.; STEEN, F.; VANHOUCKE, M. Computers & Industrial Engineering Applying a hybrid job shop procedure to a Belgian manufacturing company producing industrial wheels and castors in rubber. **Computers & Industrial Engineering**, v. 61, n. 3, p. 697–708, 2011.

SEVERINO, A.J. **Metodologia do trabalho científico**. 23. ed. São Paulo: Cortez, 2007.

SHARMA, P. Discovery of Classification Rules using Distributed Genetic Algorithm. **Procedia - Procedia Computer Science**, v. 46, n. Icict 2014, p. 276–284, 2015.

SHERAFAT, H. Algoritmos Heurísticos de Cobertura de Arcos. p. 1–174, 2004.

SHI, Y.; EBERHART, R. **Parameter selection in particle swarm optimization**. Evolutionary Programming VII – Lecture Notes in Computer Science, v. 1447, p. 591-600, 1998.

SHPARLINSKI, I. Complexity Theory. In: **Cryptographic Applications of Analytic Number Theory**. Basel: Birkhäuser Basel, p. 103–106, 2003.

SOUZA, M. F. S. **Inteligência Computacional para Otimização**. Ouro Preto. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas /InteligenciaComputacional/InteligenciaComputacional.htm>. Acesso em: 27 ago. 2014.

SUMER, E.; TURKER, M. Com puters , Environ ment and Urban System s An adaptive fuzzy-genetic algorithm approach for building detection using high-resolution satellite images. **Computers, Environment and Urban Systems**, v. 39, n. 2013, p. 48–62, 2013.

TAHERI, J.; CHOON, L. J.; ZOMAYA, A. Y.; SIEGEL, H. J. A Bee Colony based optimization approach for simultaneous job scheduling and data replication in grid environments. **Computers & Operations Research**, v. 40, n. 6, p. 1564–1578, 2011.

TAMAKI, H.; NISHIKAWA, Y. A paralleled Genetic Algorithm Basead on a Neighborhood Model and its Application to S ob shop Scheduling, **Proceedings Of the 2nd Internacional Conference on Parallel Problem Solving from Nature**, Amsterdam, 28–30 September 1992, p. 579–588.

TAPLAK, H.; ERKAYA, S.; UZMAY, İ. Sharif University of Technology Passive balancing of a rotating mechanical system using genetic algorithm. **Scientia Iranica**, v. 19, n. 6, p. 1502–1510, 2012.

TEIXEIRA, J. DE F. **MENTES E MÁQUINAS: uma introdução à ciência cognitiva**. Porto Alegre: Artes Médicas, 1998

THAMMANO, A.; PHU-ANG, A. A hybrid artificial bee colony algorithm with local search for flexible job-shop scheduling problem. **Procedia Computer Science**, v. 20, p. 96–101, 2013..

VAN WYHE, J. **Charles Darwin 1809-2009 International Journal of Biochemistry and Cell Biology**, 2009.

VON ZUBEN, F. J. Computação evolutiva: Uma abordagem pragmática. In: **Anais da I Jornada de estudos em Computação de Piracicaba e Região (1ª. JECOMP)**, Piracicaba, SP, p. 25–45, 2000.

WALKER, J. H. Cell and molecular biology concepts and experiments (3rd ed.): Karp, G. **Biochemistry and Molecular Biology Education**, v. 30, p. 274–275, 2002.

WALL, M. **GALIB: A C++ library of genetic algorithm components**. Mechanical Engineering Departament, Massachussetts Institute of Technology, 1996. Disponível em: http://lancet.mit.edu/ga/dist/galibdoc.pdf. Acessado em: 2 set. 2013.

WANG, P.; SANIN, C.; SZCZERBICKI, E. Evolutionary algorithm and decisional DNA for multiple travelling salesman problem. **Neurocomputing**, v. 150, p. 50–57, 2015.

WANG, S. F.; ZOU Y. R. Techniques for the job shop scheduling problem: a survey. Systems Engineering - **Theory & Practice**, 23, p. 49–55, 2003.

WANG, S.; LIU, M. A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem, **Computers & Operations Research**, 40, p. 1064–1075, 2013.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for search. **Technical Report SFI-TR-95-02-010**. Santa Fe: Santa Fe Institute, 1995

YAMADA, T. **Studies on methaeuristics for jobshop and flowshop scheduling problems.** 2003. 133 f. Tese (Doutorado em Informática) – Kyoto University, Kyoto, 2003.

YAMADA, T.; NAKANO, R. (1997). **Genetic Algorithms for Job Shop Scheduling Problems.** Proc. Of Modern Heuristics for Decision Suport, p. 67–81, UNICOM seminar, 18–19 March 1997, London.

YAN, M.; YE, F.; ZHANG, Y.; CAI, X.; FU, Y; YANG, X. Optimization model research on efficacy in treatment of chronic urticaria by Chinese and Western Medicine based on a genetic algorithm. **Journal of Traditional Chinese Medicine**, v. 33, n. 1, p. 60–64, 2013.

YANG, H.; YI, J.; ZHAO, J.; DONG, Z. Extreme learning machine based genetic algorithm and its application in power system economic dispatch. **Neurocomputing**, v. 102, n. 2013, p. 154–162, 2013.

YANG, J.; XU, H.; JIA, P. Neurocomputing Effective search for genetic-based machine learning systems via estimation of distribution algorithms and embedded feature reduction techniques. **Neurocomputing**, v. 113, p. 105–121, 2013.

YAS, C. Electrical Power and Energy Systems Solution to scalarized environmental economic power dispatch problem by using genetic algorithm. **International Journal of Electrical Power and Energy Systems**, v. 38, n. 2012, p. 54–62, 2011.

YING, K.-C.; LIAO, C.-J. An ant colony system for permutation flow-shop sequencing. **Computers & Operations Research**, v. 31, n. 5, p. 791–801, abr. 2004.

ZHANG, R.; CHANG, P.-C.; WU, C. A hybrid genetic algorithm for the job shop scheduling problem with practical considerations for manufacturing costs: Investigations motivated by vehicle production. **International Journal of Production Economics**, v. 145, n. 1, p. 38–52, set. 2013.

ZHANG, R.; WU, C. An Artificial Bee Colony Algorithm for the Job Shop Scheduling Problem with Random Processing Times. **Entropy**, v. 13, n. 12, p. 1708–1729, 19 set. 2011.

ZUBEN, F. J. V.; ATTUX, R. R. F. **Inteligência de Enxame**. DCA/FEEC/Unicamp e DECOM/FEEC/Unicamp, 2008.