UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

PRÓGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

BRENO DE CASTRO HONORATO E SILVA

UM ALGORITMO GENÉTICO COM CÉLULAS-TRONCO USADO NA RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO COM E SEM RESTRIÇÃO DE ESPERA

FORTALEZA – CEARÁ

2015

BRENO DE CASTRO HONORATO E SILVA

UM ALGORITMO GENÉTICO COM CÉLULAS-TRONCO USADO NA RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO COM E SEM RESTRIÇÃO DE ESPERA

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para à obtenção do título de mestre em Ciência da Computação. Área de concentração: Algoritmos e Otimização.

Orientador: Prof. Dr. Gerardo Valdísio Rodrigues Viana.

FORTALEZA – CEARÁ

2015

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Sistema de Bibliotecas

Silva, Breno de Castro Honorato e.
Um algoritmo genético com células-tronco usado na resolução do Problema de Sequenciamento com e sem restrição de espera [recurso eletrônico] / Breno de Castro Honorato e Silva. - 2015.
1 CD-ROM: il.; 4 ¾ pol.

CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico com 89 folhas, acondicionado em caixa de DVD Slim (19 x 14 cm x 7 mm).

Dissertação (mestrado acadêmico) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia, Mestrado Acadêmico em Ciência da Computação, Fortaleza, 2015.
Área de concentração: Algoritmos e Otimização.
Orientação: Prof. Dr. Gerardo Valdísio Rodrigues Viana.
Coorientação: Prof. Dr. Leonardo Sampaio Rocha.

1. Problema de Sequenciamento. 2. Algoritmo Genético. 3. Meta-heurística. I. Título.

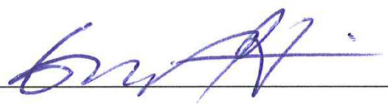
BRENO DE CASTRO HONORATO E SILVA

UM ALGORITMO GENÉTICO COM CÉLULAS-TRONCO USADO NA RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO COM E SEM RESTRIÇÃO DE ESPERA

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial para à obtenção do título de mestre em Ciência da Computação. Área de concentração: Algoritmos e Otimização.

Aprovado em 31/07/2015.

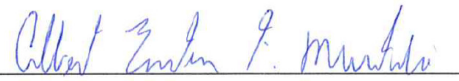
BANCA EXAMINADORA



Prof. Dr. Gerardo Valdísio Rodrigues Viana (Orientador)
Universidade estadual do Ceará - UECE

Prof. Dr. Leonardo Sampaio Rocha (Co-Orientador)
Universidade estadual do Ceará - UECE

Prof. Dr. Plácido Rogério Pinheiro
Universidade de Fortaleza - UNIFOR



Prof. Dr. Albert Einstein Fernandes Muritiba
Universidade Federal do Ceará - UFC

Dedico este trabalho à minha família, sobretudo aos meus pais que estiveram sempre presentes me apoiando e me incentivando a alcançar meus objetivos.

AGRADECIMENTOS

Em primeiro lugar, aos meus pais, por dedicarem suas vidas na construção e prosperidade desta família e pelo amor concebido aos seus filhos.

Ao meu irmão, parentes e amigos, pelo apoio e compreensão durante toda essa jornada de dedicação e restrição de tempo.

Aos Professores Dr. Gerardo Valdísio Rodrigues Viana e Dr. Leonardo Sampaio Rocha pela confiança e orientação deste trabalho.

Ao Professor Dr. José Lassance de Castro Silva pelo apoio computacional no desenvolvimento das aplicações dos algoritmos e pelos conselhos.

À todos os colegas, professores e funcionários do Programa de Mestrado Acadêmico em Ciências da Computação/UECE pela convivência harmoniosa durante todo o período do curso.

E por fim, a Deus, por ter me concedido a permissão de chegar até aqui e pelas pessoas colocadas em meu caminho ao longo desses anos.

RESUMO

Neste trabalho foi abordado o problema denominado de *Flowshop Scheduling Problem* (FSP) com e sem restrição de espera. Os problemas da classe FSP possuem grande aplicação prática em indústrias. A função objetivo a ser avaliada para os problemas foi o *makespan*. Um Algoritmo Genético (*AG*) moderno tem sido desenvolvido e aplicado na resolução dos problemas com bastante sucesso. O objetivo deste trabalho foi desenvolver um AG eficaz e eficiente para essa classe de problema e que não utilizasse inicialização eficiente e/ou hibridização com uma técnica de busca. O AG proposto levou em consideração as características de diversificação e intensificação, baseada na inspiração e criação da técnica bastante utilizada na Genética do uso de células tronco, adotada como um procedimento para o operador mutação. Foram realizados vários experimentos com as instâncias de Taillard, Reeves e Heller. Os resultados foram comparados com outros métodos encontrados na literatura onde constatou-se o bom desempenho do algoritmo proposto.

Palavras-chave: Problema de Sequenciamento. Algoritmo Genético. Meta-heurística.

ABSTRACT

This work addressed the problem called Flowshop Scheduling Problem (FSP) with and without waiting restriction. The FSP problems class have great practical application in industries. The objective function to be evaluated for the problems was the makespan. A modern Genetic Algorithm (GA) has been developed and applied in solving the problems quite successfully. The objective of this study was to develop an effective and efficient AG for this problem class and did not use efficient startup and/or hybridization with a search technique. The AG proposed here take into account the characteristics, diversification and intensification, based on the inspiration and creating widely used in genetics search as the use of stem cells, adopted as a procedure for the mutation operator. Experiments were performed with various instances of Taillard, Reeves and Heller. The results were compared with other methods in the literature where it was found the good performance of the proposed algorithm.

Keywords: Scheduling Problem. Genetic Algorithm. Metaheuristics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
AGC	Algoritmo Genético de Chen
AGM	Algoritmo Genético de Murata
AGRe	Algoritmo Genético de Reeves
AGRu	Algoritmo Genético de Ruiz
CDS	Algoritmo de Campbell, Dudek e Smith
CFSP	Continuous Flowshop Scheduling Problem
CHINS	Cheapest Insertion
FSP	Flowshop Scheduling Problem
GASA	Algoritmo Genético com Simulated Annealing
JSSP	Job Shop Scheduling Problem
IT	Insertion Technique
ISD	Iterated Steepest Descent
NEH	Algoritmo de Nawaz, Enscore e Ham
NN	Nearest Neighbor
OX	Order Crossover
PCV	Problema do Caixeiro Viajante
PMX	Partially Mapped
PO	Pesquisa Operacional
POCP	Problemas de Otimização Combinatorial Permutacional
RAL	Algoritmo de Rajedran
SD	Steepest Descent
SB2OX	Similar Block 2-Point Order Crossover
SBOX	Similar Block Order Crossover
SJOX	Similar Job Order Crossover
SJ2OX	Similar Job 2-Point Order Crossover
SP	Scheduling Problem
TS	Tabu Search
TTF	Tempo Total de Fluxo

SUMÁRIO

1 INTRODUÇÃO

Este Capítulo encontra-se dividido em quatro seções. A primeira seção contém a justificativa e relevância do trabalho. A segunda seção trata da descrição e do histórico do *Flowshop Scheduling Problem*. A terceira seção aborda os objetivos do trabalho desenvolvido. Por fim, a quarta seção apresenta o delineamento e a organização deste trabalho.

1.1 JUSTIFICATIVA E RELEVÂNCIA DO TRABALHO

A maioria das instituições enfrenta o desafio de viabilizar uma gestão integrada das informações com o processo de tomada de decisão. Um produto com um nível de serviço aceitável que contemple este objetivo, tanto na esfera operacional do curto prazo, como nos cenários táticos (de médio prazo) e estratégicos (de longo prazo) facilite e agilize as atividades por ela desenvolvidas. Neste contexto, torna-se necessário o desenvolvimento de sistemas de apoio à tomada de decisão, de resposta rápida e otimizada à problemática setorial. Um tal sistema deverá não só contribuir para a redução dos entraves existentes na instituição, mas também subsidiar políticas de decisões satisfatórias que poderão possibilitar a geração de cenários alternativos para o planejamento tático e estratégico da mesma. Também, é fundamental que o sistema possa contemplar as ações de integração entre os diversos setores das instituições, inclusive compartilhando decisões.

A Pesquisa Operacional (*PO*) apresenta uma forma otimizada de planejar, executar e acompanhar o processo de decisão. A PO surgiu durante a segunda guerra mundial e ganhou um grande número de adeptos devido os resultados satisfatórios alcançados. Ela está baseada principalmente na Matemática, Estatística e Computação, embora haja bastante aplicação em quase todas as áreas científicas do conhecimento. A presença da Pesquisa Operacional nas empresas, hoje em dia, não é mais uma questão de pesquisa, más sim de sobrevivência no mercado. Pois, no mercado global, apresentar produtos e serviços com qualidade e preço

satisfatório não é uma tarefa fácil, devido principalmente a necessidade do uso otimizado dos recursos existentes.

Neste trabalho são tratados dois problemas da Pesquisa Operacional, com suas técnicas de resolução, que podem ajudar na racionalização dos recursos para o planejamento e controle da produção de bens ou serviços de uma empresa/indústria. Em alguns casos, essas técnicas devem sofrer modificações específicas para atender as exigências (restrições) do problema específico. Vale ressaltar que, em cada problema, uma técnica sofisticada de Otimização Combinatória pode ser empregada na sua resolução. Essa técnica tem como principal objetivo apresentar boas soluções para os problemas, num tempo computacional aceitável.

Estes problemas são, geralmente, caracterizados por associarmos a cada uma de suas soluções viáveis uma permutação. Usualmente, a solução ótima de um problema de otimização combinatória permutacional consiste em se determinar dentre as (n!) soluções viáveis do problema, aquela que otimiza alguma medida de desempenho, onde n é o tamanho da instância do problema. Consequentemente, é de grande importância a existência de métodos que forneçam soluções de boa qualidade (próximas da solução ótima) em um tempo computacional razoável. As heurísticas são projetadas de modo a explorar a estrutura do problema e buscam boas soluções, não necessariamente ótimas, tanto para problemas bem definidos quanto para problemas extremamente mal estruturados, cujos objetivos e restrições podem não ser definidos explicitamente. Com base na teoria de complexidade computacional, pode-se dizer que, à medida que a dimensão desses problemas aumenta, o tempo para resolvê-los de maneira exata cresce exponencialmente.

Não é fácil desenvolver técnicas (heurísticas) que usem *processos de diversificação* e *intensificação* no espaço de soluções viáveis. No *processo de diversificação*, o objetivo é direcionar a busca para novas regiões, de forma a atingir todo o espaço de soluções possíveis, enquanto que no *processo de intensificação* há um reforço à busca na região (vizinhança) de uma solução historicamente considerada boa.

O algoritmo proposto trabalha plenamente com o processo de diversificação para os dois problemas, assim também como no processo de intensificação que depende da

característica de cada problema. O processo de diversificação se faz presente em boa parte dos algoritmos usados na resolução dos problemas de Pesquisa Operacional. Já o uso do processo de intensificação geralmente advém das boas heurísticas encontradas na literatura para cada problema específico. De acordo com os trabalhos experimentais encontrados na literatura, gerar soluções aleatórias que envolvem permutação pode concentrar ou não as buscas em um número pequeno de regiões que não atinge todo o espaço de soluções viáveis do problema. A seguir far-se-á uma abordagem sobre os problemas a serem tratados em nosso trabalho.

As empresas de manufatura enfrentam a difícil tarefa de determinar a melhor sequência de processamento de seus produtos em suas máquinas que atenda aos objetivos competitivos do negócio. A Pesquisa Operacional denomina este problema como *Scheduling Problem* (SP), ou problema de sequenciamento, e o define como: dado um conjunto de tarefas e um conjunto de máquinas, determinar uma sequência específica de tarefas que otimize uma função objetivo.

Existem vários tipos de SP, por exemplo, o *single machine scheduling problem*, *multiple machine scheduling problem* e *manpower scheduling problem*. Este trabalho trata do *multiple machine scheduling problem*, mais conhecido na literatura como *Flowshop Scheduling Problem* (FSP). O primeiro artigo publicado sobre este problema foi de Johnson (1954), que formulou e resolveu o *two-machine flowshop problem*. Segundo Gupta e Stafford Jr. (2006), de 1954 a 2004 mais de 1.200 artigos foram publicados abordando diferentes aspectos do FSP.

O FSP é definido como um fluxo unidirecional de n tarefas em m máquinas, i.e., a ordem de processamento de todas as tarefas nas m máquinas é a mesma. Considerando o caso geral do FSP o número de sequências possíveis e distintas é igual a $(n!)$, mesmo para problemas com n e m pequenos a enumeração completa de todas as soluções possíveis e distintas torna-se impraticável.

A metaheurística Algoritmo Genético (AG), baseada na evolução das espécies, tem sido aplicada com sucesso no FSP (Chen *et al.* (1995), Reeves (1995), Murata *et al.* (1996) e Ruiz *et al.* (2006)). Ruiz *et al.* (2006) desenvolveram um AG que teve um bom desempenho quando aplicado no FSP. Testar um algoritmo em instâncias conhecidas e disponíveis na

literatura é uma forma de permitir que o método possa ser comparado com outros métodos. Fink e Voβ (2003) desenvolveram várias heurísticas e alguns métodos baseados nas metaheurísticas *Simulated Annealing* e *Tabu Search*.

A primeira justificativa para a escolha do AG é a possibilidade de verificar-se que ele pode ser um excelente método de resolução tanto ao uso de recursos computacionais quanto de desempenho das soluções obtidas. A segunda justificativa é baseada na hipótese de Silva e Soma (2006) que métodos de resolução exata para problemas da classe FSP geralmente só são aplicados em problemas com até n=20 e que mesmo assim o tempo computacional ainda é muito alto, por isso a importância de desenvolver métodos que encontrem boas soluções em tempo computacional aceitável. A terceira justificativa é que se trata de uma técnica generalista, i.e., pode ser aplicada em vários problemas necessitando somente de poucas modificações. E, finalmente, a quarta justificativa é o fato de que as novas tendências aplicadas na área da Genética podem também ser adaptadas para os algoritmos evolucionários melhorarem seus desempenhos tais como os Algoritmos Genéticos.

Uma variante do FSP é o problema de sequenciamento de tarefas em máquinas sem espera na passagem das operações das tarefas, individualmente, de uma máquina à outra subsequente. Esse problema na literatura é conhecido como sendo o *Continuous Flowshop Scheduling Problem* (CFSP), ou Problema de Sequenciamento de Tarefas Contínuas, que também será abordado em nosso trabalho devido ao grande número de aplicações industriais práticas.

Portanto, neste trabalho dois problemas da classe FSP são tratados. O primeiro problema é o FSP, que assume que a seqüência de operações das tarefas processadas em cada máquina é a mesma, por isso, o número de soluções possíveis é igual a $n!$. Uma das suposições necessárias para definir o FSP (com restrição de espera) é que cada tarefa pode esperar pelo processamento entre máquinas subsequentes. Existem processos produtivos onde a suposição anterior não se aplica, i.e., as tarefas não podem parar o processamento entre máquinas consecutivas e, por isso, precisam ser processados continuamente do início ao fim, isto origina o CFSP (FSP sem restrição de espera). O segundo problema (CFSP) é abordado dado a sua importância prática e as poucas pesquisas realizadas sobre ele encontradas na literatura.

1.2 A RESOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO

A resolução de um Problema de Sequenciamento é uma operação que demanda grandes recursos, devido ao grande volume de dados inerentes a formulação do problema. Hall e Sriskandarajah (1994) afirma que existem, principalmente, três aspectos nesse contexto:

a) O tamanho do problema: tentativas de se "restringir" o espaço de busca podem ser realizadas e muitos métodos têm sido concebidos nesse sentido, contudo para que esses métodos tenham bom aproveitamento, há uma demanda de certa quantidade de informações prévias acerca do problema que, nem sempre, estão disponíveis.
b) Incertezas e a natureza dinâmica dos problemas reais: os distúrbios são inevitáveis e podem requerer apenas a substituição de um único recurso, ou eles podem exigir a completa reformulação do plano. As técnicas de otimização precisam estar aptas a enfrentar essas mudanças.
c) Inviabilidade: quando o problema não tem solução. Isso se dá, por vezes, quando as tarefas só podem ser executadas em espaços de tempo muito restritos. Existem algoritmos capazes de determinar se existe essa inviabilidade.

O mesmo autor, após executar uma série de testes com vários problemas, verificou que o AG geralmente não apresentou bom desempenho quando aplicado a classe de problemas do tipo SP, devido à própria estrutura do problema e o uso dos tempos relativos, ou seja, a modificação de um único valor afeta todas as tarefas sucessivas. Nossa proposta de trabalhar com um AG moderno é melhorar mais ainda a simplicidade de resolver o problema complexo abordado, tendo em vista que os AGs tiveram bom desempenho quando aplicados ao Problema do Caixeiro Viajante (PCV), e a relação dele com o FSP.

1.3 OBJETIVOS

O objetivo geral deste trabalho é resolver o Problema de Sequenciamento Clássico com e sem restrição de espera, bastante aplicado no meio industrial, utilizando a metaheurística Algoritmo Genético com uso de células-tronco realizando o papel do operador mutação.

Como objetivos específicos, têm-se:

a) Pesquisar o estado da arte dos dois problemas aqui abordados;
b) Pesquisar o estado da arte dos algoritmos genéticos modernos;
c) Desenvolver, implementar, testar e validar um algoritmo genético moderno aplicado no FSP, com e sem restrição de espera, onde o operador mutação é baseado no uso de células-tronco, uma técnica de grande sucesso aplicada na Genética;
d) Comparar os resultados obtidos pelo algoritmo proposto com os resultados encontrados na literatura;
e) Descrever as conclusões e descrição do trabalho científico.

Para a realização deste trabalho foi realizada um conjunto de tarefas de forma disciplinada:

a) Realizar uma revisão bibliográfica sobre os assuntos envolvidos;
b) Desenvolver, implementar e testar a técnica de solução para o problema;
c) Pesquisar e implementar a metaheurística AG nas instâncias mais significativas encontradas na literatura.
d) Comparar os resultados.
e) Escrever a versão final da dissertação.

1.4 ESTRUTURA E ORGANIZAÇÃO DO TRABALHO

A dissertação está composta por sete capítulos, organizados da seguinte forma:

Além da Introdução dada no Capítulo I, o Capítulo II mostra os conceitos mais importantes referentes à metaheurística Algoritmo Genético; a fundamentação teórica a respeito das

características do Problema de Sequenciamento, suas definições, modelo matemático e estado da arte, bem como a abordagem dos AGs já utilizados na tentativa de solução deste problema, será apresentada no Capítulo III. O Capítulo IV apresenta as principais características do Problema de Sequenciamento com restrição de espera. Enquanto no Capítulo V, o modelo de AG utilizado neste trabalho é apresentado. Já no Capítulo VI, os resultados dos testes realizados e o desempenho do algoritmo proposto na resolução do problema são abordados com destaque para o desempenho das soluções apresentadas e tempo de execução. Concluindo a dissertação, estão apresentadas no Capítulo VII, as considerações finais e sugestões para trabalhos futuros.

2 O ALGORITMO GENÉTICO

Este Capítulo encontra-se dividido em seis seções. A primeira seção apresenta um histórico sobre o AG. A segunda seção trata do modo operacional do AG. A terceira seção aborda a representação ou codificação das soluções de um AG. A estratégia geracional e a seleção do AG são apresentadas na quarta seção. A quinta seção menciona os operadores genéticos do algoritmo. Enquanto a sexta seção faz referência a parametrização e ao critério de parada do AG.

2.1 HISTÓRICO

O Algoritmo Genético foi criado por Jonh Holland durante as décadas de 1960 e 1970 (Holland, 1975), para simular computacionalmente o comportamento da seleção natural. Foi um aluno de Holland, Goldberg, o primeiro a aplicar o AG num problema de otimização, na área de projeto de gasodutos (Haupt e Haupt, 2004). Depois desta aplicação, o AG passou a ser considerado uma técnica de busca baseada nos princípios da genética e seleção natural. O AG é formado por uma população de indivíduos que representam as soluções do problema. Os indivíduos são avaliados por uma função que atribui um valor chamado aptidão a cada indivíduo da população, segundo sua qualidade em relação à função objetivo do problema. Os indivíduos são escolhidos por um procedimento inspirado na seleção natural para passarem por operações genéticas que resultam em descendentes que comporão a nova população. Estudos mostram que a nova população tem a tendência de ter indivíduos com aptidões melhores do que a população anterior (Mitchell, 1998; Haupt e Haupt, 2004). Este processo de gerar novas populações é chamado de geração. O melhor indivíduo da última população é a solução a ser apresentada para o problema.

Nos AGs, as soluções potenciais de um problema específico são codificadas em uma estrutura de dados semelhante a um cromossomo, sobre a qual são aplicados operadores de recombinação com o objetivo de preservar informações críticas. AGs têm sido aplicados em

uma grande quantidade de problemas (WHITLEY, 1993) e são definidos por Gen (2006) como técnicas estocásticas de busca baseadas nos princípios da seleção natural e da genética.

2.2 MODO DE OPERAÇÃO

Um AG baseia-se em conjuntos de soluções chamadas de populações, formadas por indivíduos, os quais são soluções codificadas. Esses indivíduos, os cromossomos, são avaliados por uma função aptidão, de forma que os melhores têm mais chance de permanecer na população ou de gerar novos indivíduos a partir de suas próprias características. Com isso, espera-se obter uma evolução dos valores de avaliação dos cromossomos ao longo do tempo e, assim, obter a melhoria das soluções. O critério de parada é a convergência das soluções para praticamente uma solução dominante ou a imposição de limites de tempo ou de iterações. Nesse momento, o individuo com a melhor avaliação representa a melhor solução encontrada para o problema tratado pelo algoritmo (KOZA, 1992; GEN, 2006).

Figura 1– Pseudocódigo de AG genérico.

```
Inicio
    gera população inicial
    avalia a população
    Enquanto o critério de parada não for atingido faça
        executa cruzamento
        executa mutação
        avalia os novos indivíduos
        seleciona os indivíduos a substituir e seus substitutos
        atualiza o critério de parada
    Fim
    apresenta a melhor solução
Fim
```

Fonte: Elaborado pelo Autor.

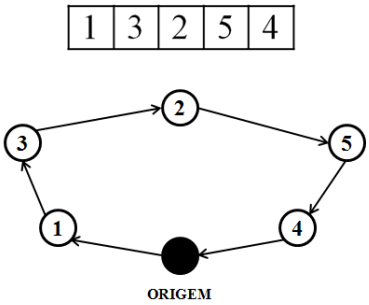
Segundo Koza (1992), os passos na preparação de um AG são a determinação da representação do cromossomo, da função de aptidão, das regras de geração da população inicial, do método de seleção, dos operadores genéticos cruzamento e mutação, da estratégia geracional, dos parâmetros e variáveis para controlar o algoritmo, do modo de

reconhecimento do resultado e do critério de parada. Na Figura 1, é apresentado o pseudocódigo de um AG genérico.

2.3 REPRESENTAÇÃO OU CODIFICAÇÃO DAS SOLUÇÕES

Baker (2003) afirma que o ponto de partida para qualquer AG é a representação das soluções da população e ressalta que, apesar de cromossomos binários terem sido favorecidos por vários estudiosos, boas implementações têm sido feitas com o uso de representações não binárias. Em um cromossomo, cada variável é um gene. Os valores possíveis de cada gene são os alelos, e a posição de cada gene é seu *lócus* (REEVES, 1993). O cromossomo representa o mapeamento de cada ponto possível no espaço de busca explorado pelo AG. Essa representação das possíveis soluções pode ser intuitiva, ou não. E, como a sua escolha pode facilitar a própria solução do problema, requer cuidado, inspiração e bom senso. Na Figura 2, é apresentado um exemplo de codificação não binária, do tipo permutacional, em que os genes correspondem a pontos de uma tarefa, e a sequência dos genes no cromossomo correspondem à sequência de processamento das tarefas nas máquinas. Esse é o tipo de representação usada neste trabalho.

Figura 2 – Exemplo de cromossomo permutacional e a sequência de processamento.



Fonte: Elaborado pelo Autor.

Holland (1975) introduz a noção de *schema* para formalizar o conceito de *building blocks*. O *schema* ou máscara é uma forma de representar configurações em que se fixam valores de determinados genes, deixando os demais em "aberto", representados por asteriscos.

A *ordem* de um *schema* é a quantidade de elementos definidos. Por exemplo, o *schema* de ordem quatro (1 2 3 * * 6), considerando o universo de permutações de 1 a 6, representa os elementos (1 2 3 4 5 6) e (1 2 3 5 4 6). Estas são instâncias desse *schema*, de forma que o mesmo define um hiperplano. Assim, o termo *schema* serve para denotar tanto *a máscara* em si quanto o conjunto de cromossomos que possuem essa máscara (MITCHELL, 1998; VIANA, 1998).

Na avaliação de uma população com *n* indivíduos, o AG estima implicitamente a aptidão média de todos os *schemas* que a compõem, variando suas participações de acordo com seus resultados. Por exemplo, os *schemas* cuja aptidão estimada fica acima da média, recebem mais números de tentativas (membros da população). Esse é o papel da seleção: achar os melhores *schemas* a cada geração. Já a avaliação simultânea e indireta dos *schemas* na população é conhecida como *paralelismo implícito*. E à medida que um AG avança em sua exploração do espaço de soluções, a estimativa da média de aptidão de um *schema* se torna mais assertiva porque o algoritmo avaliou mais instâncias desse mesmo *schema* (MITCHELL, 1998).

Por isso, é clara a importância da medida de aptidão. Seu cálculo deve ocorrer para cada indivíduo da população. Assim como a representação dos indivíduos, a escolha da medida de aptidão e da sua função geradora requer bom senso. Uma medida de desempenho dos indivíduos mal definida pode levar o AG a fazer seleções erradas, distanciando sua exploração das regiões com as soluções realmente desejadas. Além disso, a complexidade da extração do valor de aptidão de um cromossomo requer capacidade computacional, o que pode impactar no tempo de processamento do AG.

2.4 ESTRATÉGIA GERACIONAL E SELEÇÃO

A estratégia geracional de um AG é o tratamento dado para as diferentes gerações de populações durante o seu processamento. É a política de renovação e de manutenção de indivíduos de acordo com seus desempenhos, os interesses e os objetivos do AG implementado. Na forma tradicional apresentada por Holland (1975), os novos elementos criados, por cruzamento ou mutação, são inseridos em uma nova população que substituirá a

antiga. Na estratégia *steady-state3*, segundo Baker (2003), os filhos entram na população à medida que são criados e ao mesmo tempo em que são retirados os piores indivíduos, de forma que o tamanho da população permanece constante. Há também o *elitismo*, que implica na manutenção de uma determinada quantidade dos melhores indivíduos da população para a próxima geração. Mitchell (1998) ressalta a melhora significativa da performance de AGs devido ao elitismo.

Nos AGs, o modo de seleção dos indivíduos de acordo com as suas aptidões também desempenha papel fundamental. A seleção ocorre para a escolha dos indivíduos que serão preservados entre gerações ou para definir os pais que serão combinados para gerar filhos. Como exemplos de modelos de seleção, há o método da roleta, a seleção por torneios, a por *ranking* e a dinâmica. Na roleta, a chance de seleção de um indivíduo é função da sua aptidão, podendo ser linear ou não. Já no torneio, indivíduos escolhidos aleatoriamente formam grupos de tamanho fixo, e, dada uma determinada probabilidade, o melhor individuo do grupo é escolhido para cruzamento, caso contrário, qualquer um é escolhido aleatoriamente. Blickle (1997) apresenta estudos mostrando que quanto maior o tamanho do torneio, maior a pressão seletiva.

Na seleção por *ranking*, tenta-se prevenir a convergência prematura da população. Nesse tipo de seleção, os indivíduos da população são classificados de acordo com sua aptidão. E a chance de seleção do indivíduo é função da sua classificação. Dessa forma, indivíduos em posição imediatamente superior ou inferior na classificação têm praticamente as mesmas chances independentemente dos valores absolutos da medida de aptidão. Com isso, mantém-se a pressão alta quando a variabilidade é baixa e a reduz na situação oposta. Este racional também é a base dos processos de seleção dinâmicos, do tipo Boltzmann, segundo os quais, em momentos distintos do processamento, são necessárias pressões distintas de seleção.

2.5 OPERADORES GENÉTICOS E HIBRIDIZAÇÃO

O papel dos operadores genéticos em um AG é criar novos indivíduos a partir de cromossomos existentes na população. Existem operadores de cruzamento e de mutação. No

primeiro caso, considerado por Mitchell (1998) e Gen (2006) o principal operador genético, são criados cromossomos filhos pela recombinação das características dos cromossomos pais, os quais são escolhidos conforme o método de seleção vigente.

Já as mutações são operadores de retaguarda que produzem mudanças aleatórias e espontâneas em vários cromossomos. Seu papel, normalmente, é pequeno nos AGs, pois estes tomam como base os efeitos criativos da recombinação cromossômica propiciada pelas operações de cruzamento e a exploração dos efeitos do princípio *darwiniano* da sobrevivência dos mais fortes (KOZA, 1992). Ainda assim, as mutações podem atuar tanto na diversificação quanto na intensificação da busca pelo espaço de soluções independendo do mecanismo codificado.

Mitchell (1998) expõe a ideia de Holland (1975) na qual a adaptação é a tensão entre a busca pelo novo e a exploração desse novo encontrado. A autora explica que a tensão surge porque qualquer movimento no sentido de explorar novas áreas do espaço de soluções, experimentando novos *schema s*ão testar instâncias cuja aptidão são baixas e desvia a exploração de *schemas* já testados e com bons resultados. Ela afirma também que o sistema tem que continuar testando novas possibilidades, mas também tem que incorporar e usar continuamente a experiência passada como guia para o comportamento futuro, pois um balanceamento ótimo entre explorar o certo ou não explorar o incerto deve ser encontrado. De certa forma, também por causa dessa "indecisão" e do seu caráter generalista, diz-se que os AGs realizam buscas superficiais e por isso é conveniente hibridizá-los com outras heurísticas.

Com a abordagem híbrida, técnicas de otimização local são aplicadas a cada novo filho da população. Nessas situações, os AGs são usados para explorar a população enquanto os métodos heurísticos são usados para explorar as soluções. Por causa dessas características complementares dos AGs e das heurísticas convencionais, o método híbrido normalmente se sai melhor que cada qual individualmente (GEN, 2006).

Blum e Roli (2003) reforçam esse conceito ao afirmarem que a maior parte dos casos de sucesso com modelos evolucionários utilizam procedimentos de busca local. Para estes autores, as razões se tornam aparentes ao se analisar as forças de cada método: métodos

baseados em populações são melhores em identificar áreas promissoras no espaço de soluções, enquanto heurísticas de busca local são boas em explorar essas áreas. Essas heurísticas partem de uma solução provavelmente razoável ou boa e trabalham de forma orientada ao problema, refinando o cromossomo trabalhado de uma forma que o AG não faria. No Capítulo 2, Seção 2.5, vários exemplos do operador cruzamento foram dados.

2.6 CRITÉRIO DE PARADA E OUTROS PARÂMETROS

Em se tratando de um método não exato, no qual se busca ótimo sem necessariamente encontrá-lo, o AG pode seguir avaliando e gerando indivíduos indefinidamente. Por isso, é preciso estabelecer um critério de parada para interromper a busca e apresentar um resultado, normalmente a melhor solução encontrada até a interrupção do processamento. O número de gerações, o tempo de processamento e a diversidade da população são os critérios mais comuns de encerramento de execução de um AG.

Juntamente com o critério de parada e seu valor, outros parâmetros também devem ser estabelecidos para a execução de um AG. Koza (1992) coloca o tamanho da população e o número máximo de gerações como os principais parâmetros de controle de um AG, enquanto os secundários são as taxas de reprodução e mutação. E, dependendo do modelo e da implementação, pode haver outros parâmetros.

Apesar de existirem estudos nesse sentido, Mitchell (1998) não acredita na existência de uma formulação geral de parametrização por causa da variedade de problemas, codificações e outras especificidades possíveis em diferentes aplicações. A autora afirma ainda que o tamanho ótimo de uma população, as taxas de cruzamento e de mutação mudam ao longo do processamento.

Através da revisão bibliográfica foram escolhidos oito componentes como sendo os mais importantes num projeto de AG:

1. Escolha da representação do cromossomo;
2. Definição da função de aptidão (função objetivo do problema);

3. Definição da população inicial;
4. Escolha do método de seleção;
5. Escolha dos operadores genéticos cruzamento e mutação;
6. Escolha da estratégia geracional;
7. Escolha do critério de parada; e
8. Escolha dos valores dos parâmetros (calibragem do AG).

No Capítulo V, mais adiante, será mostrado como se deu o desenvolvimento do AG proposto e como é feita sua aplicação no FSP e CFSP. A única diferença da aplicação de um problema para o outro dar-se-á na função *fitness* que cada problema usa para avaliar uma solução, dada pela permutação de *n* tarefas. A seguir, serão descritas as características do FSP.

3 O PROBLEMA DE SEQUENCIAMENTO

Neste capítulo, a definição e a modelagem matemática do FSP são apresentadas na primeira e segunda seção, respectivamente. Na terceira seção, um método de resolução do FSP é apresentado dentro da otimização combinatória permutacional. O estado da arte do FSP está descrito na quarta seção. Enquanto a quinta seção aborda os AGs aplicados no FSP.

3.1 DEFINIÇÃO DO FSP

Antes de definir o FSP é importante esclarecer que *flowshop* não é sinônimo de linha de montagem, mesmo que a característica do *flowshop* seja um fluxo que pareça ser constante de trabalhos através de um conjunto de máquinas em série, conforme Gupta e Stafford Jr.(2006). A seguir são apresentadas três diferenças entre estes dois tipos de modelo de sistema de produção.

a) No ambiente *flowshop* existe uma variedade de produtos e na linha de montagem existe um produto padrão;

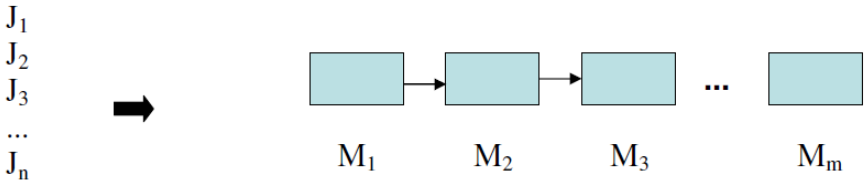
b) No ambiente *flowshop* as tarefas não são obrigadas a passarem em todas as máquinas dependendo das necessidades tecnológicas e na linha de montagem todas as tarefas têm que passar por todas as estações de trabalho; e

c) No ambiente *flowshop* cada tarefa tem seu próprio tempo de processamento em cada máquina e na linha de montagem todas as unidades dos produtos têm o mesmo tempo padrão em cada estação de trabalho.

Assim o FSP pode ser definido como sendo um conjunto de n tarefas J_1, J_2, ..., J_n, onde cada tarefa deve ser processada em m máquinas M_1, M_2, ..., M_m. Cada tarefa demanda m operações, com uma operação representando o tempo de processamento da tarefa por máquina. As tarefas seguem o mesmo fluxo de operações nas máquinas, i.e., para qualquer j=1, 2, ..., n, a tarefa Jj deve ser processada primeira na máquina M_1, depois na máquina M_2, e assim por diante até a última máquina, no caso a máquina M_m, conforme mostra a Figura 3,

dada abaixo. Caso a tarefa Jj não utilize todas as máquinas, o seu fluxo continua sendo o mesmo, todavia com o tempo de operação sendo igual a zero para as máquinas que ela não vai utilizar. Uma máquina pode processar somente uma operação de cada vez, e iniciada uma operação, ela deve ser processada até a sua conclusão. O número de sequências distintas possíveis para realização das tarefas nas máquinas é da ordem $O(n!)$. O problema consiste em realizar todas as n tarefas no menor tempo possível.

Figura 3 – Ilustração do FSP.



Fonte: Elaborado pelo Autor.

Conforme Silva e Soma (2006) um *input* do FSP é dado por n, m e uma matriz P(n x m) de elementos não negativos, onde P_{ij} denota o tempo de processamento da tarefa J_j na máquina M_i. Seguindo os 4 parâmetros da notação A/B/C/D adotada por Conway *et al.*(1967), o problema é classificado como n/m/P/Fmax. Na menos antiga notação paramétrica V/W/Y, proposta por Graham *et al.* (1979), o problema é denotado como sendo F/prmu/Cmax.

O FSP pertence à classe dos problemas NP-completo, no sentido forte, quando m for maior que 3, conforme Garey e Johnson (1979). No caso em que m for menor ou igual a 3, o problema pode ser solucionado em tempo polinomial. A seguir são descritas algumas suposições da aplicação prática do FSP.

Suposições relacionadas às tarefas:

T1 – Cada tarefa é liberada para a fábrica no começo do período de programação.
T2 – Cada tarefa pode ter sua própria data de entrega fixa e não sujeita a mudança.
T3 – Cada tarefa é independente das demais.
T4 – Cada tarefa consiste de operações específicas que são realizadas por somente uma máquina.

T5 – Cada tarefa tem uma sequência tecnológica pré-estabelecida fixa e que é igual para todas as demais tarefas.

T6 – Cada operação de uma tarefa requer um tempo de processamento finito e conhecido para ser processada nas várias máquinas. Nesse tempo de processamento estão incluídos tempos de transporte, *setup* e outros. O tempo de processamento é independente dos tempos de processamento das tarefas anteriores e posteriores.

T7 – Cada tarefa é processada não mais que uma vez em cada máquina.

T8 **–** Cada tarefa pode esperar entre máquinas consecutivas, ou seja, estoque em processo permitido.

Suposições em relação às máquinas:

M1 – Cada setor é composto de somente uma máquina e a fábrica tem somente uma máquina de cada tipo.

M2 – Cada máquina está inicialmente desocupada no início do período de programação.

M3 – Cada máquina na fábrica opera independentemente das outras e, por isso, pode operar na taxa de produção máxima.

M4 – Cada máquina só pode processar uma tarefa por vez.

M5 – Cada máquina está continuamente disponível para processar tarefas durante período de programação e não há interrupções devido a quebras, manutenção ou outras causas.

Suposições relacionadas às políticas de operação:

1 – Cada tarefa é processada tão logo seja possível. Por isso, não há intenção de fazer a tarefa ficar esperando ou fazer a máquina ficar ociosa.

2 – Cada tarefa é considerada uma entidade individual mesmo que possa ser composta por um conjunto de unidades.

3 – Cada tarefa, uma vez iniciada, é processada até o fim, ou seja, o cancelamento de tarefas não é permitido.

4 – Cada operação de uma tarefa, uma vez iniciada numa máquina, é completada antes que outra tarefa possa começar na mesma máquina, ou seja, nenhuma preempção é permitida.

5 – Cada máquina processa as tarefas na mesma ordem.

É importante conhecer estas suposições já que os outros problemas da classe FSP foram criados a partir de alguma modificação nelas. Este é o caso do *sequence dependent setup time Flowshop Scheduling Problem* que foi criado a partir da alteração da suposição *6* que deixa de considerar o tempo de *setup* como fazendo parte do tempo de processamento e passa a considerar os dois tempos separadamente. O exemplo 01 mostra a aplicação do problema cujo modelo matemático é dado a seguir..

Exemplo 01: Queremos minimizar o tempo de processamento de 3 tarefas a serem processadas em 2 máquinas, sendo que todas as tarefas iniciam primeiro na máquina 1 e depois na máquina 2. Os tempos de processamento das tarefas nas máquinas são dados na Tabela 1 abaixo.

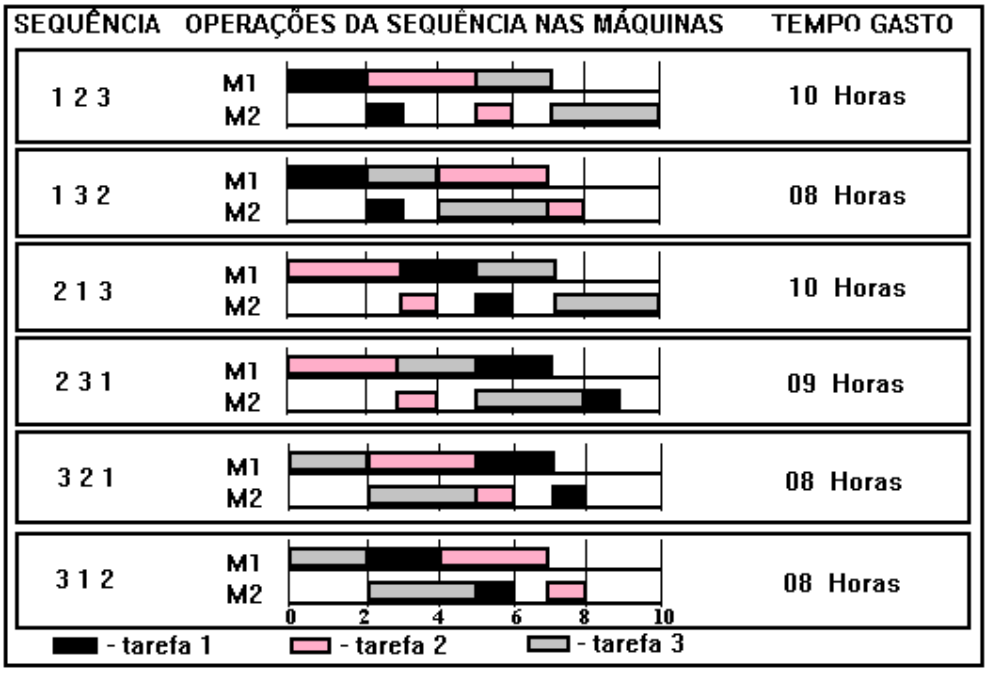
Tabela 1 - Tempo de operacionalização das tarefas nas máquinas, dado em horas.

	Tarefa 1	**Tarefa 2**	**Tarefa 3**
Máquina 1	02	03	02
Máquina 2	01	01	03

Fonte: Elaborado pelo Autor.

Quer-se então determinar qual a seqüência de processamento das 3 tarefas nas 2 máquinas com o menor tempo possível. Com este objetivo, a medida de desempenho usado neste caso é denominado, na literatura, de *makespan*. Existem somente 6 tipos distintos de ordenar as tarefas nas máquinas: 123, 132, 213, 231, 312, e 321. A Figura 4, dada a seguir, mostra o tempo gasto por cada uma das seqüências (soluções do problema) e as devidas alocações de processamento nas 2 máquinas. Conclui-se que as sequências 132, 312 e 321 são as soluções ótimas do problema, com um tempo gasto de 8 horas para processar as 3 tarefas. Se uma determinada técnica de resolução aproximada do problema tivesse apresentado a seqüência 123, com tempo gasto de 10 horas, haveria um desvio de 25% da solução ótima. Isto equivale a duas horas a mais das seqüências ótimas.

Figura 4 – Ilustração das soluções do exemplo 01.



Fonte: Elaborado pelo Autor.

3.2 UM MODELO MATEMÁTICO PARA O FSP

A seguir são apresentados a notação necessária e um modelo matemático em Programação Linear Inteira e Mista para o FSP.

Notação:

a) T_{ij} é uma variável do modelo que representa o tempo para iniciar o processamento da tarefa j na máquina i, com $1 \leq j \leq n$ e $1 \leq i \leq m$;

b) X_{ijk}, é uma variável binária definida por:

$$X_{ijk} = \begin{cases} 1, \text{ se a tarefa } j \text{ precede a tarefa } k \text{ na máquina } i. \\ 0, \text{ caso contrário.} \end{cases}$$

Com $i = 1, 2, 3, ..., m$, $j = 1, 2, 3, ..., (n-1)$ e $k = (j+1), (j+2), ..., n$.

c) W é um número bastante grande;

Observações:

i) São dados do problema: n, m e a matriz P;

ii) Em (b), j e k não variam de 1 até n, porque seriam comparadas as tarefas j e k na mesma máquina duas vezes; e

iii) As variáveis do tipo X_{ijk} estabelecem a sequência de processamento das tarefas em cada uma das m máquinas.

$$\text{Minimizar } Z = \sum_{j=1}^{n} T_{mj} \qquad 3.1$$

sujeito a:

$$T_{rj} \geq T_{r-1,j} + p_{r-1,j}\,, \ \forall \ r = 2,3,...,m \text{ e } j = 1, 2,..., n. \qquad 3.2$$

$$\left.\begin{array}{l} T_{ik} \geq T_{ij} + p_{ij} - W * (1 - X_{ijk}) \\ T_{ij} \geq T_{ik} + p_{ik} - W * X_{ijk} \\ x_{ijk} \in \{0,1\} \end{array}\right\} \quad \forall \ 1 \leq i \leq m, j = 1, 2,..., n-1 \text{ e } k = j+1, j+2,...n \qquad \begin{array}{l} 3.3 \\ 3.4 \\ 3.5 \end{array}$$

$$T_{ij} \geq 0, \ \forall \ 1 \leq i \leq m \text{ e } 1 \leq j \leq n \qquad 3.6$$

Onde:

a) A função objetivo 3.1 tenta minimizar a soma para a conclusão das n tarefas nas m máquinas através da obtenção do menor tempo para iniciar cada tarefa na última máquina. Quanto mais cedo as tarefas forem sendo processadas na última máquina mais cedo será o tempo gasto para a conclusão de todas as tarefas. Para o caso da tarefa j não utilizar a máquina m substitui-se T_{mj} por T_{rj},onde r é a última máquina a ser utilizada pela tarefa j;

b) O grupo de restrições 3.2 representa o fluxo que as tarefas devem seguir para serem concluídas. Cada equação do tipo 3.2 determina que a (i+1)-*ésima* operação da tarefa j não pode iniciar até que a *i-ésima* operação da tarefa j na máquina i seja concluída.

c) Os grupos de restrições 3.3 e 3.4 comparam a relação de precedência das *n* tarefas nas *m* máquinas. Estes grupos de restrições também não permitem que uma máquina processe duas tarefas ao mesmo tempo;

d) As restrições do grupo 3.5 representam o atendimento à definição da variável X_{ijk} como sendo binária;

e) As restrições do grupo 3.6 definem a não-negatividade dos tempos de processamento;

f) O valor de *W* faz com que uma das restrições do grupo 3.3 se torne coerente com a definição da variável *Xijk* da seguinte forma:

i) Se $X_{ijk} = 1$, então 3.3 fica $T_{ik} \geq T_{ij} + p_{ij}$, coerente com a definição de X_{ijk}, enquanto isso 3.4 fica $T_{ij} \geq T_{ik} + p_{ik} - W$, ou seja, T_{ij} fica maior ou igual que um número negativo, logo $T_{ij} \geq T_{ik} + p_{ik} - W$ se torna verdadeiro devido ao grupo de restrição do tipo 3.6.

ii) Se $X_{ijk} = 0$, então 3.4 fica $T_{ij} \geq T_{ik} + p_{ik}$, coerente com a definição de X_{ijk},enquanto isso 3.3 fica $T_{ik} \geq T_{ij} + p_{ij} - W$, ou seja, T_{ik} maior ou igual que um número negativo, logo $T_{ik} \geq T_{ij} + p_{ij} - W$ se torna verdadeiro devido ao grupo de restrições do tipo 3.6.

iii) Podemos adotar *W* como sendo 1000*Max $\{p_{ij}\}$, $\forall\ 1 \leq i \leq m$ e $1 \leq j \leq n$.

Complexidade do modelo:

a) Variáveis do tipo X_{ijk}: $m \times n \times (n - 1) / 2$;

b) Variáveis do tipo T_{ij}: $m \times n$;

c) Restrições do grupo 3.2: $n \times (m - 1)$;

d) Restrições do grupo 3.3 e 3.4: $m \times n \times (n - 1)$;

e) Número total de variáveis: $m \times n \times (n + 1) / 2$; e

f) Número total de restrições: $n \times (m \times n - 1)$.

3.3 UM MODELO COMBINATORIAL PERMUTACIONAL DO FSP

Um Problema de Otimização Combinatória Permutacional (*POCP*) pode ser definido por um terno (*S*, *g*, *n*), onde *S* é o conjunto de todas as soluções do problema, *g* é uma função ou procedimento que aplica a cada solução viável $s \in S$ um número real e *n* é uma instância do problema. O número de soluções existentes para um POCP é representado por |S| (cardinalidade de *S*) e igual a *n*! (fatorial de *n*). O objetivo é encontrar uma solução $s^* \in S$ que otimize um dado critério de desempenho representado pela função ou procedimento *g*. Representa-se *s* como uma permutação de *n* elementos, ou seja, $s = <a_1, a_2, ..., a_n>$, com $a_i \neq a_j$, tal que $1 \leq i, j \leq n$ e $i \neq j$.

Segundo Silva e Soma (2006) o FSP pode ser modelado como um POCP da seguinte forma:

a) Um elemento $s = <J_1, J_2, ..., J_n>$ do conjunto de soluções viáveis *S* é representado por uma permutação das *n* tarefas, com a ordem de *s* determinando a sequência na qual as tarefas serão processadas; e

b) O procedimento *g*, dado anteriormente na Figura 5, determina o valor do tempo gasto (g) para processar as *n* tarefas dada pela sequência *s*, mais precisamente tem-se que g é o tempo utilizado no processamento da última tarefa de *s* na última máquina M_m.

Para determinar a sequência *s* com menor valor de *g* seria necessário enumerar e avaliar todas as *n*! sequências distintas de *S*. A Tabela 2, abaixo, mostra para alguns valores de *n* a quantidade de soluções distintas de *S* e o tempo computacional, caso o tempo de processamento do procedimento *g* para cada sequência fosse igual 0,001 segundos. Os resultados da Tabela 2 demonstram que para valores de *n* maiores que 20 fica inviável a enumeração completa de todas as soluções.

Figura 5 – Procedimento para calcular o tempo gasto pela sequência *s*.

Entrada: *m*, *n*, permutação *s*, Matriz *P*(*m* x *n*).
Saída: *g* (tempo gasto para processar todas as *n* tarefas usando a seqüência *s*)

```
for(i=1; i<=m; i++)
  for(j=1; j<=n; j++)  t[i][j]=0;
for(i=2; i<=m; i++) t[i][ s[1] ]=t[i-1][ s[1] ]+ p[i-1][ s[1] ];
for(j=2; j<=n; j++)
  for(i=1; i<=m; i++)
  {  if (i==1) {  t[1][ s[j] ]=t[1][ s[j-1] ]+p[1][ s[j-1] ];  }
     else {
       y=t[i][ s[j-1] ]+p[i][ s[j-1] ];   /* tempo da tarefa anterior na maquina atual */
       x=t[i-1][ s[j] ]+p[i-1][ s[j] ];   /* tempo da maquina anterior, para tarefa atual */
       if (x>=y) t[i][ s[j] ]=x;  else  t[i][ s[j] ]=y;
     }
  }
g=t[m][ s[n] ] + p[m][ s[n] ];
```

* O elemento t_{ij} representa o tempo para iniciar a tarefa J_j na máquina M_i.

Fonte: Elaborado pelo Autor.

Tabela 2 – Tempo de processamento do número de soluções de *S*.

n	Total de Soluções	Tempo Computacional
5	1,20 x 10^2	0,12 seg
10	3,63 x 10^6	3.628,80 seg
13	6,2 x 10^9	72 dias
14	8,7 x 10^{10}	2,76 anos
15	1,3 x 10^{12}	41,4 anos
20	2,43 x 10^{18}	7,71 x 10^7 anos

Fonte: Elaborado pelo Autor.

3.4 ESTADO DA ARTE DO FSP

A análise do histórico do progresso das técnicas utilizadas na resolução dos problemas da classe FSP serve para situar o método proposto neste trabalho. Gupta e Stafford Jr. (2006) analisaram o desenvolvimento da pesquisa em relação ao FSP desde o trabalho de

Johnson (1954) até 2004. Esse período foi dividido em cinco décadas (1955-1964, 1965-1974, 1974-1984, 1985-1994 e 1995-2004) e para cada período as suposições, as formulações para o problema e as abordagens de solução foram analisadas.

A primeira década tratou o FSP principalmente do ponto de vista teórico. Além da formulação de Johnson (1954) para duas máquinas foi desenvolvido o *m-machine flowshop* para a minimização do *makespan* (quando o FSP tem como medida de desempenho o procedimento *g* dado anteriormente na Figura 2). Foram desenvolvidas poucas técnicas para a solução do FSP. As duas técnicas que mais se destacaram foram à programação matemática (Wagner, 1959; Manne, 1960) e a simulação de Monte Carlo (Sisson, 1959; Muth eThompson, 1963). O tamanho dos problemas resolvidos era pequeno por três motivos: i) falta de capacidade computacional; ii) falta de eficientes programas de computador; e iii) a maioria das variações do *two-machine flowshop problem* é NP-Difícil.

A segunda década apresentou um maior número de técnicas de solução e outras funções objetivo além do *makespan*. Os primeiros a propor a abordagem combinatorial foram Dudek e Teuton (1964). A técnica *branch and bound* para o FSP foi desenvolvida por Lomnicki (1965). Nessa época também começou o desenvolvimento das primeiras heurísticas para encontrar boas soluções para o FSP de grandes dimensões.

Na terceira década, com a publicação da teoria da NP-Completude por Garey e Johnson (1979), a pesquisa em relação ao FSP passou a ter duas direções. Uma direção vai na tentativa de identificar a complexidade de vários FSP (Brucker, 1998; Lawler*et al.* 1993) e a outra no desenvolvimento de novas heurísticas. Nessa década também ocorreu a proposição de variações do FSP como o que separa o tempo de *setup* do tempo de processamento, que considera a data de entrega na função objetivo e que considera o tempo de processamento estocástico.

Na quarta década surgiu o *hybrid flowshop* que consiste em cada centro de trabalho poder ser constituído de múltiplas máquinas em paralelo. Nessa década iniciou-se o uso das metaheurísticas (Aarts e Lenstra, 1997): *Tabu Search*; *Simulated Annealing;* e Algoritmo Genético. Também foram desenvolvidas técnicas baseadas em inteligência artificial, sistemas de apoio à decisão e sistemas especialistas (sistemas que utilizam o conhecimento empírico

acumulado da resolução de problemas que já ocorreram para ajudar a resolução de novos problemas).

Na quinta década continuou o crescimento na criação de novos problemas, funções objetivo e abordagens de resolução. A principal novidade foi o aumento das pesquisas considerando funções multiobjetivo (T'Kindt e Billaut, 1993).

3.5 ALGORITMOS GENÉTICOS PARA A RESOLUÇÃO DO FSP

Nesta Seção far-se-á a apresentação do estudo realizado sobre o estado da arte na resolução do FSP através da técnica Algoritmo Genético. Quatro Algoritmos Genéticos são apresentados, que durante muito tempo tiveram bom desempenho na resolução do problema específico.

3.5.1 AG de Chen *et al.* (1995)

Chen *et al.* (1995) desenvolveram um AG para o FSP com o *makespan* como critério de desempenho. O AG foi testado em problemas cujos dados foram extraídos de sequências de números gerados de maneira pseudoaleatória. O AG desenvolvido é composto das seguintes partes:

a) Representação dos indivíduos;

b) Geração da população inicial e tamanho da população;

c) Avaliação da aptidão e método de seleção;

d) Operadores genéticos; e

e) Critério de parada.

A representação genética adotada foi a permutacional. Por exemplo, para uma instância com $n = 8$, o indivíduo pode ser representado por qualquer sequência de oito tarefas como sendo s=<2 1 7 8 5 3 6 4>.

A população inicial é gerada a partir dos métodos CDS, desenvolvido por Campbell *et al.*(1970), e de Dannenbring (1977). Os (m – 1) primeiros membros da população são gerados pelo método CDS, o elemento de número m é gerado pelo método Dannenbring, e a geração do elemento m + 1 até o último elemento da população é determinado a partir do primeiro elemento da população, segundo e demais elementos mediante a troca de posições de duas tarefas escolhidas aleatoriamente. Segundo *Chen et al.* (1995), depois de vários experimentos que não foram explicitados no artigo, conclui-se que 60 indivíduos é o melhor tamanho para a população.

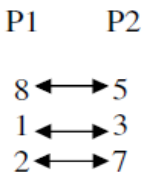
A forma como é calculada a aptidão de cada indivíduo é descrita a seguir. O primeiro passo é calcular o valor do *makespan* de todos os indivíduos da população. O segundo passo é selecionar o *C*máx que é o *makespan* do maior valor da população. O terceiro passo calcula a aptidão que é igual a diferença entre o valor do *makespan* do indivíduo e o *C*máx. O método de seleção não foi explicitado, a única informação dada foi que a seleção é baseada na aptidão do indivíduo.

O operador de *crossover* utilizado foi o *Partially Mapped* (PMX) desenvolvido por Goldberg (1989). A seguir são apresentados os procedimentos do operador PMX juntamente com uma ilustração para n = 8.

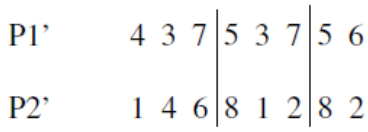
1 – Escolher aleatoriamente um intervalo comum aos dois pais. Por exemplo, as posições de quatro a seis.

P1	4	3	7	8	1	2	5	6
P2	1	4	6	5	3	7	8	2

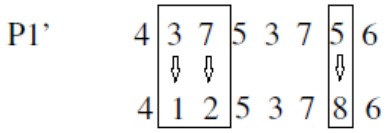
2 – Armazenar relacionadamente os elementos dos dois intervalos selecionados. Na ilustração o armazenamento relacionado é o seguinte:

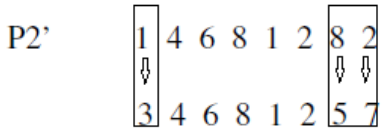


3 – Trocar os dois intervalos. Na maioria das vezes os indivíduos resultantes não são viáveis porque ocorrem tarefas repetidas. O passo 4 corrige esse problema.



4 – Trocar os elementos repetidos e que não estão dentro dos intervalos selecionados pelas tarefas armazenadas de P1 relacionados às tarefas de P2 e vice-versa. Depois desse passo os indivíduos são totalmente viáveis.





Chen *et al.* (1995) realizaram testes com problemas gerados aleatoriamente para determinar a melhor combinação para as taxas de *crossover* e mutação. Foram usadas para a taxa de *crossover* os valores 1, 0.95 e 0.90 e para a taxa de mutação os valores 0.01, 0.005 e 0. O resultado do experimento mostrou que a melhor combinação foi uma taxa de *crossover* igual a 1 e uma taxa de mutação igual a 0. O significado destes valores é que sempre os indivíduos escolhidos para reprodução passam pelo processo de *crossover* e nunca um indivíduo da população sofre mutação.

O critério de parada do método foi o número de gerações. Após alguns experimentos Chen *et al.* (1995) chegaram à conclusão que depois de 20 gerações a população do AG estagnava e não havia mais melhoria.

3.5.2 AG de Reeves (1995)

Reeves (1995) desenvolveu um AG para o FSP com o *makespan* sendo o critério de desempenho. O AG tem como principal diferença uma probabilidade de mutação adaptativa e foi testado nas instâncias desenvolvidas pelo próprio autor e de Taillard (1993). A representação genética utilizada foi a permutacional que é sempre a opção natural para este tipo de problema. A aptidão de cada indivíduo na população é igual a *vmáx – v*, onde *vmáx* é o valor do maior *makespan* da população e *v* é o valor do *makespan* do indivíduo.

A seleção do método de geração da população inicial deu-se com dois experimentos: i) gerada aleatoriamente; e ii) com um indivíduo gerado pela heurística NEH de Nawaz *et al.*(1983) e o restante da população gerada aleatoriamente. O segundo método obteve soluções tão boas quanto o primeiro método, mas com um tempo computacional menor, por isso, foi o método selecionado.

O método de seleção é composto por dois tipos de seleção. O primeiro pai é selecionado usando o tipo de seleção por *ranking* com probabilidade *P*i dada pela equação abaixo e o segundo pai é selecionado com probabilidade uniforme de acordo com a aptidão.

$$P_i = \frac{k}{M\ x\ (M+1)}$$

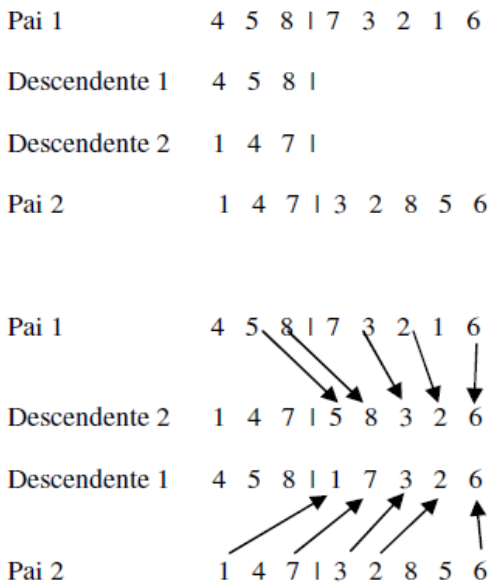
onde:

- i: é um indivíduo da população, i = 1, 2, ..., M;

- k: é a posição do indivíduo na população em ordem descendente com relação ao *makespan;* e

- M: é o tamanho da população.

Figura 6 – Operador de cruzamento *one-point crossover.*



Fonte: Elaborado pelo Autor.

Foram utilizados operadores genéticos de *crossover* e mutação. O operador de *crossover* foi o *one-point crossover* que consiste em escolher um mesmo ponto de corte em cada um dos pais, copiar as tarefas à esquerda do ponto de corte de cada pai para cada um dos descendentes e copiar as tarefas que faltam do outro pai na mesma ordem relativa. A Figura 6, dada anteriormente, ilustra o funcionamento do *one-point crossover*.

O operador de mutação utilizado foi o *shift* que consiste em escolher uma tarefa aleatoriamente e colocar numa posição da sequência escolhida aleatoriamente. Também é utilizada uma estratégia geracional que consiste em inserir os novos indivíduos no lugar dos indivíduos com aptidão menor que a média da aptidão da população. Esta estratégia garante a sobrevivência dos indivíduos com melhor aptidão, mas por outro lado, diminui a diversidade da população. O critério de parada utilizada foi o tempo de execução, dada a facilidade no momento de realizar os experimentos para comparar com outros métodos.

Durante os experimentos preliminares Reeves (1995) notou que a população convergia prematuramente. Por isso implementou uma probabilidade de mutação *P*m adaptativa. Um parâmetro *D* é estabelecido para controlar a diversidade da população. A

razão *v*min/*v*med que mede a diversidade da população é calculada ao fim de cada geração, onde *v*min é o valor do menor *makespan* da população e *v*med é o valor do *makespan* médio da população. Se está razão é maior ou igual a *D*, a probabilidade de mutação é multiplicada por um fator de decréscimo θ (0<θ<1), caso contrário esta probabilidade retorna ao valor inicial *P*mini.A probabilidade *P*mini é alta no início da busca e diminui durante o processo de evolução. Ela retorna a crescer quando a diversidade da população está baixa.

Os valores dos parâmetros usados por Reeves (1995) foram:

- Tamanho da população (*M*) : 30;
- Probabilidade de *crossover* (*P*c) : 100 %;
- Probabilidade inicial de mutação (*P*mini) : 80 %;
- Taxa de decréscimo da probabilidade de mutação: 0,99; e
- Parâmetro de controle de diversidade (*D*) : 0,95.

Reeves (1995) também adotou a estratégia de fazer todos os pais passarem pelo processo de *crossover*, já que usou uma taxa de 100%.

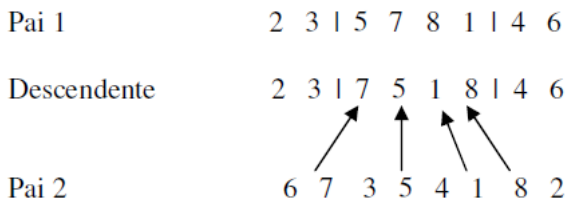
3.5.3 AG de Murata *et al.* (1996)

Murata *et al.* (1996) desenvolveram três tipos de estudos com AG para o FSP com o *makespan* sendo o critério de desempenho. Foram realizados estudos para os operadores genéticos, os valores dos parâmetros e as opções de hibridização. Neste AG, permutações foram usadas para representar as soluções do problema.

Foram testadas duas formas para calcular a probabilidade de seleção, sendo uma delas escolhida porque conseguiu a maior pressão de seleção, e assim obteve os melhores resultados. Quanto aos operadores genéticos de *crossover* e mutação, foram testados dez operadores de *crossover* para determinar o melhor para o FSP. *O two-point crossover* (versão 1) foi o que obteve o melhor desempenho: dois pontos da sequência são escolhidos aleatoriamente de um dos pais. As tarefas que ficam desses pontos para as extremidades são

copiados para o descendente. As tarefas que faltam na sequência do descendente são copiadas na mesma ordem relativa do outro pai como mostra a Figura 7. Foram testados quatro operadores de mutação para determinar o melhor para o FSP. A mutação *shift* foi a que obteve o melhor desempenho.

Figura 7 – *Two-point crossover* (versão 1).



Fonte: Elaborado pelo Autor.

Estudos também foram realizados para quantificar os parâmetros do AG e a melhor combinação foi tamanho da população *Npop* = 10, taxa de *crossover Pc* = 1.0 e taxa de mutação *Pm* = 1.0. Estes valores significam que todos os indivíduos da população são substituídos por indivíduos gerados no processo de *crossover* e todos os indivíduos da população sofrem mutação. O melhor indivíduo da população anterior é copiado para a nova população no lugar de um indivíduo escolhido aleatoriamente.

O último estudo realizado foi testar qual o melhor método para hibridizar com o AG. Foram testadas duas opções, um algoritmo *simulated annealing*e e um algoritmo de busca local. A hibridização do AG com o algoritmo de busca local foi o que obteve o melhor desempenho. Para reduzir o custo computacional da busca local usou-se a estratégia de avaliar só uma parte α da vizinhança de uma solução, por exemplo, α = 10% significa que 10% das soluções da vizinhança são escolhidas aleatoriamente. Não foi mencionada que tipo de estrutura de vizinhança utilizou-se. Para verificar qual seria o melhor valor para o parâmetro α, foram feitos testes com os seguintes valores: 100%, 75%, 50%, 10% e 5%. O melhor valor para α foi 75%.

O AG com busca local (α = 75%) é o melhor algoritmo para o FSP desenvolvido por Murata *et al.* (1996). Ele é composto de sete passos que são descritos a seguir.

1 – Inicialização : gera uma população inicial de indivíduos de forma aleatória de tamanho *N*pop.

2 – Busca local : aplica a busca local em todos os indivíduos da população se um critério de parada é satisfeito a busca é encerrada senão o processo continua e as novas soluções comporão a população atual. O critério de parada é o seguinte: se depois dos α vizinhos avaliados tiver havido melhoria em algum indivíduo a busca é encerrada, senão, são avaliados todos os vizinhos de todos os indivíduos da população.

3 – Seleção : seleciona *N*pop pares de pais da população atual de acordo com a probabilidade de seleção.

4 – *Crossover:* Aplica o operador *crossover* a cada par de pais escolhidos com uma probabilidade *P*c. Se o operador não for aplicado é escolhido um dos pais para compor a nova população.

5 – Mutação: aplica o operador de mutação a cada indivíduo com a probabilidade *P*m, esta probabilidade é referente a cada indivíduo e não a cada tarefa como na representação binária.

6 – Estratégia elitista: adiciona o indivíduo de melhor aptidão da população atual na nova população no lugar de um indivíduo escolhido aleatoriamente.

7 – Critério de parada: finaliza a execução do algoritmo se a condição de parada (tempo de execução) é satisfeita, caso contrário, retorna ao passo 2.

3.5.4 AG de Ruiz *et al.* (2006)

Ruiz *et al.* (2006) desenvolveram dois novos AG para o FSP sendo também o *makespan* o critério de desempenho.Os AGs têm: inicialização eficiente; estratégia geracional que só aceita indivíduos melhores e com sequência única; quatro novos operadores de *crossover* que foram desenvolvidos; um procedimento para evitar a convergência prematura; e proposta de uma busca local para a hibridização com o primeiro AG. A representação da

solução utilizada foi permutacional, onde a ordem relativa das tarefas na permutação indica a ordem de processamento das mesmas.

Para gerar a população inicial Ruiz *et al.* (2006) desenvolveram uma modificação na heurística NEH de Nawaz *et al.* (1983). A modificação foi a seguinte: depois de ordenar todas as tarefas em ordem decrescente do tempo total de processamento, são escolhidas duas tarefas aleatoriamente e colocadas nas duas primeiras posições, depois disso o procedimento continua igual ao NEH original. A população inicial é composta por um indivíduo gerado pela heurística NEH, (*B*i% - 1) indivíduos gerados pela heurística NEH modificada e os (100 – *B*i) % indivíduos restantes são gerados aleatoriamente. O parâmetro *B*i indica o percentual de indivíduos gerados eficientemente.

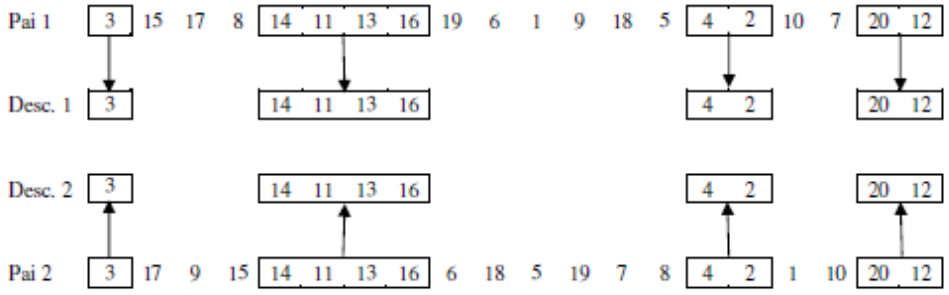
Duas condições foram desenvolvidas para controlar como os indivíduos gerados substituem os indivíduos da população atual. A primeira é que um indivíduo gerado só substitui o indivíduo da população atual com o pior *makespan* se o seu *makespan* for menor, e a segunda condição é que a sequência do novo indivíduo seja única em relação à população atual. Isto ajuda a manter a diversidade na população.

Ruiz *et al.* (2006) desenvolveram quatro novos operadores de *crossover* para o FSP que são baseados na ideia de identificar e manter os bons blocos construídos. O primeiro operador foi chamado de *Similar JobOrder Crossover* (SJOX) que funciona da seguinte maneira. Os dois pais são examinados posição por posição. Quando as tarefas são idênticas na mesma posição elas são copiadas para os dois descendentes (Figura 8), depois são escolhidos um ponto de corte aleatoriamente e cada um dos descendentes herda todas as tarefas a esquerda do ponto de corte de um dos pais (Figura 9) e finalmente as tarefas que faltam em um dos descendentes são copiados em ordem relativa do outro pai (Figura 10).

O segundo operador de *crossover* foi resultado da constatação de que algumas vezes muitas tarefas iguais isoladas apareciam, por isso, desenvolveram outro operador de *crossover* chamado *Similar BlockOrder Crossover* (SBOX). A única diferença do SBOX em relação ao SJOX é que no primeiro passo só são copiados blocos idênticos de ao menos duas tarefas. O terceiro operador de *crossover* é chamado de *Similar Jobs 2-Point Order Crossover* (SJ2OX) que é similar ao operador SJOX com a diferença que são dois pontos de corte, ao invés de um.

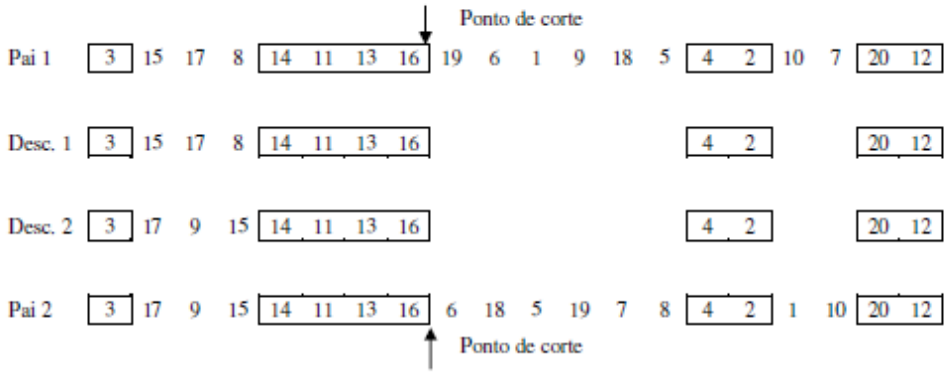
As tarefas entre dois pontos de corte são copiadas de um dos pais, enquanto os trabalhos dos dois extremos são preenchidos em ordem relativa pelas tarefas dos extremos do outro pai. O quarto operador de *crossover* é chamado de *Similar Block 2-Point Order Crossover* (SB2OX) que é similar ao operador SBOX só que utiliza dois pontos de corte.

Figura 8 – Primeiro passo do *crossover* SJOX.



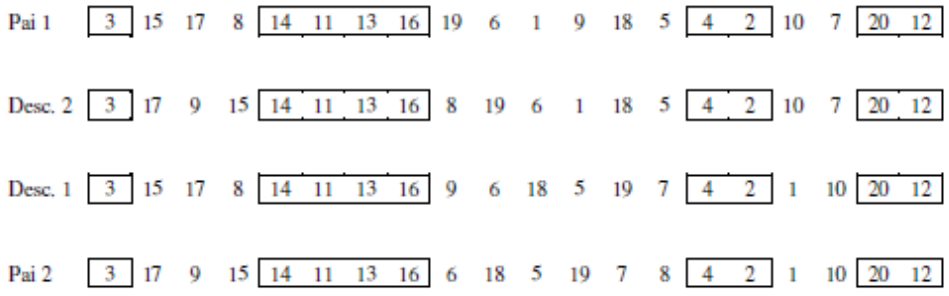
Fonte: Ruiz *et al.* (2006)

Figura 9 – Segundo passo do *crossover* SJOX.



Fonte: Ruiz *et al.* (2006)

Figura 10 – Terceiro passo do *crossover* SJOX.



Fonte: Ruiz *et al.* (2006)

O tipo de mutação implementada foi a *shift* que consiste em escolher uma tarefa aleatoriamente para ser colocado numa posição escolhida também aleatoriamente. O procedimento *restartscheme* foi desenvolvido com o objetivo de evitar a convergência prematura do AG. Este procedimento é executado toda vez que um número de gerações sucessivas *G*r são executados e não é gerado um indivíduo melhor. Este procedimento é descrito a seguir.

1 - Colocar a população em ordem crescente em relação ao *makespan*;

2 - Manter os 20% dos melhores indivíduos;

3 - Gerar 40% de novos indivíduos a partir da mutação *shift* dos 20% melhores indivíduos;

4 - Gerar 20% dos indivíduos a partir da modificação da heurística NEH; e

5 - Gerar os 20% restantes dos indivíduos de forma aleatória.

A hibridização consiste da aplicação de uma busca local baseada na técnica *insertion neighborhood* que realiza todas as possíveis inserções e armazena a melhor sequência. Se o resultado for melhor do que a sequência atual a busca é repetida, senão a busca é encerrada. A busca local é realizada a cada geração em cada indivíduo com a probabilidade *P*enh. O primeiro AG não tem a etapa de busca local, ou seja, *P*enh = 0 e o segundo AG tem a etapa de hibridização, ou seja, *P*enh > 0.

O projeto de experimentos consiste da comparação de todas as possíveis combinações de operadores genéticos e valores dos parâmetros. O total de combinações avaliadas foram 15.360. Criou-se 68 instâncias combinando os valores de n e m, n = {20, 50, 80, ..., 440, 470, 500} e m = {5,10, 15, 20}. O resultado interessante foi que em comparação com os limites inferiores dos problemas gerados, a melhor combinação de operadores e valores dos parâmetros obteve um desvio de 3,22%, enquanto a pior combinação obteve 3,85%. Como a diferença não foi tão significativa Ruiz *et al.* (2006) afirmaram que isto se deveu a robustez dos AGs desenvolvidos. A seguir, serão descritas as características do CFSP.

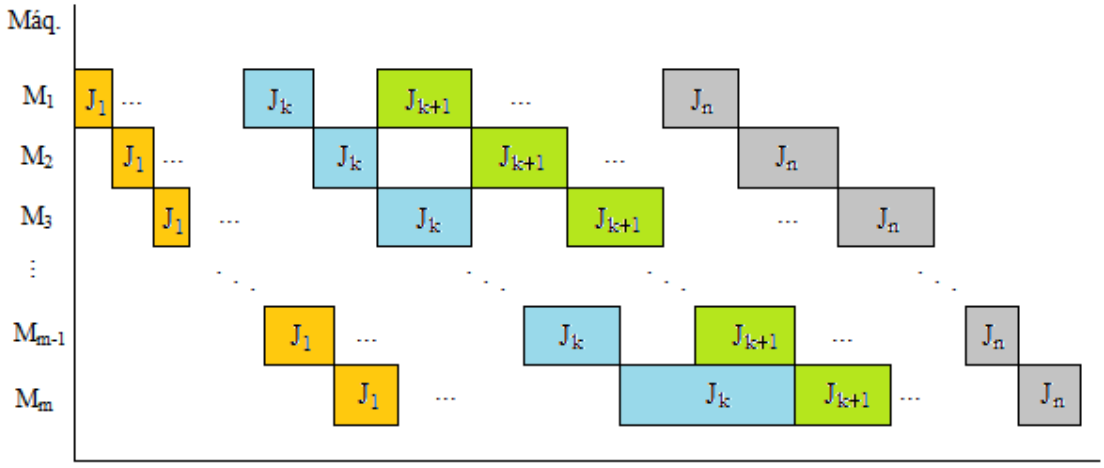
4 O FSP SEM RESTRIÇÃO DE ESPERA

Este capítulo contempla o FSP sem a restrição de espera, denominado na literatura de *Continuos Flowshop Scheduling Problema*. Na primeira seção é feita a definição do CFSP. Na segunda seção, o problema é tratado através da otimização combinatória permutacional. A terceira seção mostra alguns tipos de algoritmos de resolução para o problema CFSP.

4.1 DEFINIÇÃO DO CFSP

O CFSP é um problema da classe FSP originado por pelo menos uma alteração na suposição *T8*, dada na Seção 3.1. A suposição *T8* permite que as tarefas esperem entre máquinas consecutivas pelo processamento. A definição do CFSP é semelhante ao do FSP com o acréscimo da restrição de que as tarefas não podem esperar entre máquinas consecutivas.

Figura 11 – Gráfico de Gantt de um CFSP com *n* tarefas e *m* máquinas.



Fonte: Elaborado pelo Autor.

Dado um conjunto de *n* tarefas para serem processados num conjunto de *m* máquinas, onde todas as tarefas usam a mesma ordem de processamento nas máquinas e

depois de iniciada uma tarefa, ela não deve esperar por processamento entre duas máquinas consecutivas, i.e., as tarefas devem ser processadas continuamente, o tempo de processamento da tarefa *j* na máquina *i* é dado por p_{ij}, $i = 1, 2, 3,..., m$ e $j = 1, 2, 3,..., n$. O CFSP consiste em determinar uma seqüência específica das *n* tarefas que otimize um critério de desempenho estabelecido. Esta definição só é válida na prática se as demais suposições apresentadas na Seção 3.1 forem verdadeiras. A Figura 11 apresenta o gráfico de Gantt de um CFSP. Nota-se no gráfico de Gantt que não existe folga entre o processamento individual de cada uma das tarefas.

4.2 O CFSP COMO UM POCP

O modelo matemático para o CFSP não foi apresentado neste capítulo devido sua semelhança com o modelo matemático descrito para o FSP, apresentado na seção 3.2, com a alteração apenas das restrições do grupo 2 que devem ser de igualdade.

Também é pequena a diferença do modelo POCP para o CFSP em relação ao FSP. Esta diferença está somente no cálculo da função *g*. Fink e Voβ (2003) apresentam uma fórmula para calcular o valor da função *g*. A seguir apresenta-se a Equação 4.1 para calcular o valor de *g* de uma permutação *s* com critério de desempenho sendo o *makespan*.

$$Makespan: \quad g_{CFSP} = \sum_{j=2}^{n} d_{s(j-1),s(j)} + \sum_{i=1}^{m} p_{iS(n)} \qquad 4.1$$

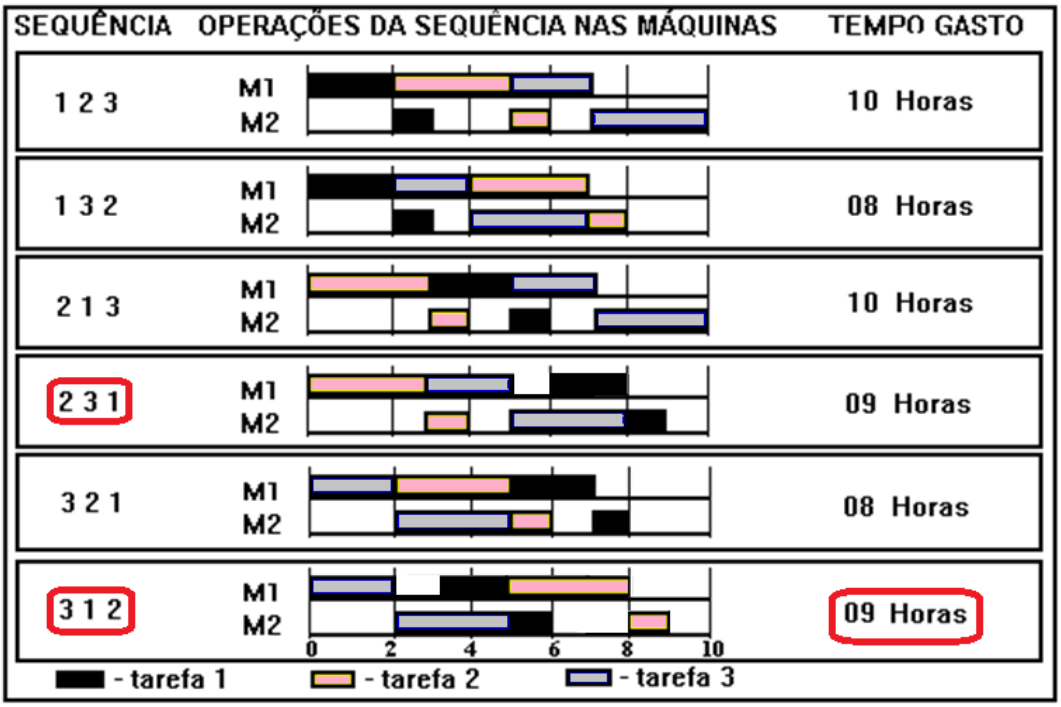
A incógnita d_{jk} consiste no tempo de espera na primeira máquina entre o começo da tarefa *j* e o começo da tarefa *k*, quando *k* é processado posteriormente a *j*, em que $1 \leq j \leq n$ e $1 \leq k \leq n$, com $j \neq k$. A Equação 4.2 mostra como se calcula o valor de d_{jk}.

$$d_{jk} = \max_{1 \leq i \leq m} \{ \sum_{h=1}^{i} p_{hj} - \sum_{h=2}^{i} p_{h-1,k} \} \qquad 4.2$$

Segundo Röck (1984) os primeiros a analisarem a complexidade do CFSP para m>2 foram Lenstra *et al.* (1977), seguidos por Papadimitriou e Kanellakis (1980) em que ambos provaram que o problema é NP-*hard* em sentido forte para m≥4. Existem casos que podem

ser tratados polinomialmente, desde que os tempos de processamento satisfaçam uma estrutura especial dada por Panwalkar e Woollam (1980). Por fim, Röck (1984) provou para m≥3 que o CFSP com o critério de minimização sendo o *makespan* é NP-*hard* em sentido forte.

Figura 12 – Ilustração da resolução do CFSP.



Fonte: Elaborado pelo Autor.

Tomando os dados do Exemplo 01, descrito no Capítulo III, como sendo uma instância do CFSP tem-se que determinar qual a sequência de processamento das 3 tarefas nas 2 máquinas com o menor tempo possível, levando em consideração que as tarefas não podem ficar esperando o seu processamento ao passar de uma máquina para outra (CFSP). Com este objetivo, a medida de desempenho usada neste caso é também denominada, na literatura, de *makespan*. Existem somente 6 tipos distintos de ordenar as tarefas nas máquinas: 123, 132, 213, 231, 312, e 321. A Figura 12, dada anteriormente, mostra o tempo gasto por cada uma das seqüências (soluções do problema) e as devidas alocações de processamento nas 2 máquinas. Conclui-se que as sequências 132 e 321 são as soluções ótimas do problema, com

um tempo gasto de 8 horas para processar as 3 tarefas. Na Figura 12, destaques são dados para as sequências 231, com alocação diferenciada das tarefas nas máquinas comparada com o FSP, e 312, que teve aumento no tempo gasto de 1 hora quando comparada com a mesma sequência aplicada no FSP.

4.3 MÉTODOS DE RESOLUÇÃO PARA O CFSP

Foram analisados quatro artigos com propostas de métodos para a resolução do CFSP. Dois desses artigos usaram as instâncias de Reeves (1995) e Heller (1960) para testar os métodos propostos, que também serão usadas por nós para compararmos o nosso método de resolução.

4.3.1 AG e *Simulated Annealing* de Aldowaisan e Allahverdi (2003)

Aldowaisan e Allahverdi (2003) desenvolveram quatro algoritmos de busca local para o CFSP com o *makespan* como critério de desempenho. Dois desses algoritmos têm a solução básica inicial obtida pelo algoritmo *simulated annealing* (SA) de Chakravarthy e Rajendran (1999) e os outros dois têm a solução básica inicial obtida pelo AG de Chen *et al.* (1996). Para o processo de busca local foi desenvolvida uma nova heurística chamada de *Insertion Technique* (IT). Os algoritmos foram testados com problemas gerados aleatoriamente. O método IT foi inspirado na heurística NEH criada por Nawaz *et al.* (1983). O método IT consiste em considerar duas tarefas consecutivas como um bloco e o inserir em todas as posições ainda disponíveis na sequência. A Figura 13, dada a seguir, apresenta a descrição do método IT.

Segundo Aldowaisan e Allahverdi (2003) até aquele momento a metaheurística *simulated annealing* ainda não tinha sido aplicada ao CFSP. Eles optaram por adaptar o algoritmo SA desenvolvido por Chakravarthy e Rajendran (1999) para outro tipo de problema de *scheduling*. O AG usado por Aldowaisan e Allahverdi (2003) é o mesmo desenvolvido por Chen *et al.* (1996) para o CFSP.

Figura 13 – Descrição do método IT.

Passo 1: Escolher uma seqüência *s*, onde os elementos são representados por *s* (j), j é a posição na seqüência e varia de 1 a *n*. *k* := 0.

Passo 2: *k* := *k* + 1. Selecionar *s* (*k*) e *s* (*k* + 1) para formar o bloco. Colocar o bloco nas posições de *k* a *n*. Para cada seqüência criada, trocar as posições de *s* (*k*) e *s* (*k* + 1) dentro do bloco e calcular o valor do *makespan*. Selecionar para a seqüência corrente a com menor *makespan*.

Passo 3: Repetir o passo 2 até *k* = *n* - 1.

Fonte: Nawaz *et al.* (1983).

Em Aldowaisan e Allahverdi (2003) encontram-se detalhes dos pseudocódigos das buscas locais SA-1, GEN-1, SA-2 e GEN-2. As buscas locais SA-1 e GEN-1 têm as soluções básicas iniciais obtidas pelos algoritmos SA e GEN, respectivamente. A busca local é implementada através da aplicação dos métodos NEH e IT alternadamente cinco vezes cada. Segundo os autores, os experimentos realizados mostraram que dessa forma a qualidade da solução final era melhor que se os métodos fossem aplicados sucessivamente. Os experimentos também mostraram que a aplicação desses métodos mais de cinco vezes não proporcionava melhoria significativa na qualidade da solução final. As buscas locais SA-2 e GEN-2 têm as soluções básicas iniciais obtidas pelos algoritmos SA-1 e GEN-1, respectivamente. A busca local foi implementada através da aplicação do procedimento *pairwise* três vezes. O procedimento *pairwise* consiste em examinar cada possível troca de pares de uma tarefa numa posição com todas as outras tarefas. Os experimentos também mostraram que não havia melhoria significativa na qualidade da solução final quando o procedimento *pairwise* era aplicado mais de três vezes.

Aldowaisan e Allahverdi (2003) testaram seus quatro algoritmos, o SA de Chakravarthy e Rajendran (1999), o AG de Chen *et al.* (1996) denominado de GEN, a melhor heurística desenvolvida por Gangadharan e Rajendran (1993) denominado de GAN-RAJ e a heurística desenvolvida por Rajendran (1994) denominada de RAJ. Para os testes foram

usados 1.000 problemas gerados aleatoriamente com 20 combinações diferentes de 40 a 120 tarefas e 5 a 20 máquinas. Os algoritmos foram implementados em linguagem FORTRAN e os testes realizados num SUN SPARC Station 20. Os tempos de execução foram omitidos, mas segundo os autores o maior tempo de execução foi 10 segundos. O resumo dos resultados dos testes foi apresentado no artigo, onde vê-se que os algoritmos propostos (SA-1, SA-2, GEN-1 e GEN-2) têm melhores desempenhos que os demais comparados, já que as soluções iniciais são geradas a partir de heurísticas eficientes. O desempenho na média é semelhante para: SA-1 e GEN-1; e SA-2 e GEN-2. As instâncias usadas por eles não foram disponibilizadas para efeito de comparações.

4.3.2 As Heurísticas de Aldowaisan e Allahverdi (2004)

Aldowaisan e Allahverdi (2004) desenvolveram também oito heurísticas para o CFSP com o *makespan* como critério de desempenho. As heurísticas diferem em três aspectos: primeiro, a escolha entre os dois métodos de inserção; segundo, a escolha entre os dois critérios de parada; e finalmente, em usar ou não o procedimento de troca *pairwise*. As heurísticas propostas são comparadas às duas heurísticas de Rajendran e Chaudhuri (1990) e ao algoritmo genético de Chen *et al.* (1996). As heurísticas foram testadas com problemas também gerados aleatoriamente.

As heurísticas propostas por Aldowaisan e Allahverdi (2004) denotadas por PH*i* (*proposed heuristic*), onde *i*= 1, 2, 3 e 4, fazem uso da seqüência gerada pelo algoritmo ASI (algoritmo de seqüência inicial), descrito no artigo com detalhes, onde as duas primeiras heurísticas propostas se diferenciam apenas pelo método de inserção. A primeira heurística, PH_1, usa o método de inserção NEH, desenvolvido por Nawaz *et al.* (1983). A segunda heurística, PH_2, usa o método de inserção, RAZ, proposto por Rajendran e Ziegler (1997) *apud* Aldowaisan e Allahverdi (2004). Os dois próximos métodos PH_3 e PH_4 se diferenciam entre si pelo método de inserção e em relação aos dois primeiros métodos dados, pelo procedimento de parada. A finalização nas duas primeiras heurísticas ocorre quando $r > 10$ e nos dois próximos métodos quando $r > 10$ ou $k = 2$.

A partir da incorporação do procedimento de troca *pairwise* às heurísticas anteriores, obtêm-se novas heurísticas denotadas por PH*i*(p), onde i= 1, 2, 3 e 4. O procedimento *pairwise* consiste em examinar cada possível troca de pares de uma tarefa numa posição com todas as outras tarefas.

Entre as oito heurísticas desenvolvidas por Aldowaisan e Allahverdi (2004) foram apresentados os resultados dos testes das heurísticas PH_1, $PH_{1(p)}$, PH_3, $PH_{3(p)}$, PH_4 e $PH_{4(p)}$ e comparadas com as duas heurísticas de Rajendran e Chaudhuri (1990) denominadas de R-C1 e R-C2 e o AGChen desenvolvido por Chen *et al.* (1996). Os experimentos computacionais foram realizados com 750 problemas, gerados aleatoriamente com 5 combinações diferentes de 50 a 400 tarefas e 5 a 25 máquinas com 30 replicações em cada classe. Os algoritmos foram implementados em linguagem FORTRAN e os experimentos realizados num SUN SPARC Station 20. Os tempos de execução em segundos foram apresentados no artigo. Novamente, essas instâncias também não foram disponibilizadas na internet. Através do artigo, percebe-se que os tempos usados pelos métodos de Aldowaisan e Allahverdi (2004) foram muito maiores que os tempos usados pelos métodos comparados. Vê-se que a heurística $PH_{1(p)}$ foi a que teve o melhor desempenho, devido à qualidade da solução inicial obtida pelo método PH_1, que entre os métodos sem a etapa de melhoria foi o que obteve o melhor desempenho. A conclusão de Aldowaisan e Allahverdi (2004) foi que seus métodos são melhores que as heurísticas existentes para o CFSP, entretanto foram usados tempos de execução muito grandes.

4.3.3 GASA de Shuster e Framinan (2003)

Wang e Zeng (2001) desenvolveram um método híbrido chamado de GASA para resolver o *Job Shop Scheduling Problem* (JSSP). O GASA é uma combinação das técnicas *Genetic Algorithms* (GA) e *Simulated Annealing* (SA). Schuster e Framinan (2003) adaptaram o GASA para o CFSP com o critério de desempenho sendo o *makespan*, e usaram as instâncias de Reeves (1995) e Heller (1960).

Wang e Zheng (2001) criaram um novo operador de *crossover* para ser usado no GASA. Neste operador primeiramente um conjunto {1, 2, ..., n} é dividido em dois sub-

conjuntos A_1 e A_2 aleatoriamente, sendo que cada subconjunto tem que possuir ao menos um elemento. Cada elemento contido num subconjunto é copiado para um descendente na mesma posição que ocupava no indivíduo pai. Depois são escolhidos aleatoriamente dois indivíduos da população atual para serem os pais s_1 e s_2. Os descendentes s'_1 e s'_2 são criados da seguinte forma: s'_1 herda os elementos de s_1 pertencentes a A_1 e os elementos de s_2 pertencentes a A_2; s'_2 herda os elementos de s_1 pertencentes a A_2 e os elementos de s_2 pertencentes a A_1.

O GASA inicia com uma população inicial de tamanho P_{size} gerada aleatoriamente, uma temperatura inicial t_0 e o critério de parada é o número L de iterações sem melhoria. A cada iteração os procedimentos de *crossover*, mutação e SA são aplicados. Estes três procedimentos estão bem detalhados no artigo de Wang e Zheng (2001).

Para comparar o desempenho do GASA, Schuster e Framinan (2003) o testaram nas instâncias de Reeves (1995) e Heller (1960) e compararam com os resultados obtidos pela heurística RAJ de Rajendran (1994). O GASA foi implementado em linguagem C++ e os experimentos foram realizados num computador Athlon 1.400 MHz. Os resultados dos testes e os tempos utilizados pelo GASA foram apresentados no artigo, onde se verifica que o GASA obtém melhores resultados do que o método RAJ, só que para isso precisa usar uma grande quantidade de tempo de execução. Além disso, para as instâncias com maior número de tarefas e máquinas o GASA fica abaixo do método RAJ.

4.3.4 Os Algoritmos de Grabowski e Pempera (2005)

Grabowski e Pempera (2005) propuseram cinco algoritmos de busca local, dois deles baseados na técnica *Descending Search* (DS) e três baseados na metaheurística *Tabu Search* (TS), para resolver o CFSP com o *makespan* sendo o critério de desempenho. As características mais importantes desses algoritmos são: o emprego de multimovimento e lista tabu dinâmica. A solução inicial de todos os algoritmos é obtida pelo método NEH de Nawaz *et al.* (1983). As instâncias de Reeves (1995) e Heller (1960) foram utilizadas nos experimentos computacionais para avaliar o desempenho dos métodos.

O tipo de movimento e a estrutura de vizinhança são componentes importantes dos algoritmos propostos. Um movimento é definido pelo par v = (x, y) que são duas posições da permutação *s*, com x, y $\in \{1, 2, ..., n\}$ e x $\neq$ y. Segundo Grabowski e Pempera (2005), baseados na literatura e em experimentos realizados, o movimento *shift* é o que melhor se adapta ao CFSP, por isso, é adotado pelos algoritmos. A vizinhança da permutação *s* consiste das permutações s_v obtidas pela execução de todos os movimentos de um dado conjunto de movimentos Z e denotada por $N(Z, s) = \{ s_v \mid v \in Z\}$. Os algoritmos propostos geram vizinhanças através de movimentos $Z = \{ (x,y) \mid x, y \in \{1, 2, ..., n\}, y \notin \{x, x-1\}\}$ de cardinalidade $(n - 1)^2$, onde a condição $y \notin \{x, x-1\}$ evita a redundância de movimentos.

Para acelerar a convergência às boas soluções, os algoritmos utilizam um procedimento chamado multimovimento, que tem o propósito de guiar a busca para regiões mais promissoras onde boas soluções podem ser encontradas. O multimovimento consiste de um conjunto de vários movimentos individuais que são executados simultaneamente numa única iteração do algoritmo. A execução do multimovimento gera permutações que diferem significativamente daquelas obtidas pela execução de um único movimento e conduz o processo de busca para regiões até o momento não visitadas do espaço de soluções. Segundo Grabowski e Pempera (2005) a aplicação de multimovimento em algoritmos de busca local é uma forma de adotar as estratégias de intensificação e diversificação no processo de busca. O multimovimento é composto de um conjunto de movimentos individuais proveitosos e independentes.

O subconjunto $PZ = \{ v \in Z \mid C_{max}(s_v) < C_{max}(s)\}$ é chamado de conjunto de movimentos proveitosos e contêm todos os movimentos do conjunto Z que geram permutações s_v com menores *makespan* que s. Dois movimentos $v_1 = (x_1, y_1) \in PZ$ e $v_2 = (x_2, y_2) \in PZ$ são chamados independentes em relação a permutação s se cada uma das posições x_1 e y_1 estão separadas de cada uma das posições x_2 e y_2 por pelo menos uma tarefa.

O primeiro algoritmo proposto, no artigo, foi um DS que consiste em pesquisar a vizinhança N até encontrar um movimento $v^* \in Z$ que gere uma permutação $s_{v^*} \in N$ com menor *makespan* que *s*, uma solução inicial. Dessa forma a permutação s_{v^*} se torna a nova solução, i.e., $s := s_{v^*}$ e o algoritmo é repetido até que nenhuma permutação melhor seja encontrada. O segundo algoritmo proposto é uma combinação de DS e multimovimento

(DS+M). O DS+M começa de uma solução inicial *s* e uma vizinhança N(Z, *s*). Para a vizinhança N o conjunto de multimovimentos V' é criado de acordo com o procedimento descrito anteriormente. O multimovimento V' é realizado e a permutação resultante $s_{v'}$ se torna a nova solução, i.e., $s := s_{v'}$, o algoritmo é repetido até que V' = ϕ.

Segundo Grabowski e Pempera (2005) a metaheurística *Tabu Search* (TS) não tinha sido aplicada até então no CFSP. Assim, foram desenvolvidos três algoritmos baseados em TS que têm como principal característica à utilização de lista tabu dinâmica com o propósito de evitar que o processo de busca fique preso a um ótimo local. O comprimento da lista T é alterado quando o número de iterações (*iter*) do TS atinge um valor específico chamado de *pick*. Esse tipo de lista foi empregado primeiramente no *very fast* TS, proposto por Grabowski e Wodecki (2004).

Em relação a permutação *s*, um movimento (x, y) $\in$ Z é proibido: se A(s(x)) $\cap$ {s(x+1), s(x+2), ..., s(y)} $\neq \phi$, se x < y; ou B(s(x)) $\cap$ {s(y), s(y+1), ..., s(x-1)} $\neq \phi$ se x > y. Onde: A(j) = { $i \in$ J | (j, i) $\in$ T} e B(j) = { $i \in$ J | (i, j) $\in$ T}. O conjunto A(j) (ou B(j)) indica quais tarefas são processadas depois (ou antes) da tarefa *j* em relação ao conteúdo atual da lista tabu T.

O TS começa de uma solução básica inicial *s* a qual é aplicada à vizinhança N(Z, *s*). Primeiramente, o melhor movimento v* $\in$ Z que gera a permutação $s_{v^*} \in$ N(Z, *s*) com o menor *makespan* é escolhido. Se $C_{max}(s_{v^*})$ < C*, então o movimento v* é selecionado para o processo de busca. Caso contrário ($C_{max}(s_{v^*}) \geq$ C*), então é criado o conjunto *UZ* de movimentos não proibidos (*UF*) que não tem o status tabu e definido como *UZ* = { v $\in$ Z | movimento v é *UF*}. No próximo passo, $s_{v^*} \in$ N(*UZ*, s) com o menor *makespan* é escolhido para o processo de busca. Se o movimento v* é selecionado, então o par de tarefas correspondentes ao movimento v* é adicionado à lista tabu T e a permutação resultante s_{v^*} é criada. No passo seguinte, a permutação se torna a nova solução, i.e., $s := s_{v^*}$ e o algoritmo começa a próxima iteração. Se todos os movimentos de Z são proibidos, um caso muito raro, i.e., *UZ* = ϕ, então o elemento mais velho da lista tabu T é retirado dela e a busca é repetida até que um movimento *UF* possa ser encontrado.

O quarto algoritmo proposto é uma combinação de TS e multimovimento (TS+M). O algoritmo TS+M é similar ao TS exceto que em cada iteração um multimovimento V', que contém vários movimentos simples, é realizado, ao contrário de um movimento simples v*. Se numa iteração do TS+M, o multimovimento V' contêm não mais que um movimento, i.e., $|V'| \leq 1$, então o TS+M se transforma em TS.

O quinto e último algoritmo é uma combinação de TS e multimovimento (TS+MP) que é realizado somente em situações específicas. O multimovimento é realizado a cada vez que um número de iterações (*Piter*) onde não ocorre melhoria no *makespan* é atingido. Se *Piter* é um número suficientemente grande o multimovimento nunca será criado e TS+MP se transforma em TS. O parâmetro *Piter* foi calibrado experimentalmente. Para comparar o desempenho dos seus algoritmos, Grabowski e Pempera (2005) realizaram testes nas instâncias de Reeves (1995) e Heller (1960) e compararam com os resultados obtidos pela heurística RAJ de Rajendran (1994) e o GASA de Shuster e Framinan (2003). Os algoritmos foram implementados em linguagem C++ e os experimentos foram realizados em um Pentium 1000. O resumo dos resultados dos experimentos computacionais é apresentado a seguir na Tabela 3, onde verifica-se que o melhor desempenho foi obtido pelo algoritmo TS-M. A seguir, serão descritas as características do AG aplicado ao FSP e CFSP.

Tabela 3 – Resumo dos resultados dos experimentos de Grabowski e Pempera (2005).

Métodos	**Desvio (%)**	**Tempo (s)**
RAJ	0,00	-
GASA	-1,18	200,13
DS	-4,51	0,02
DS-M	-4,53	0,00
TS	-6,50	0,86
TS-M	-6,59	0,87
TS-MP	-6,56	0,87

Fonte: Grabowski e Pempera (2005).

5 MÉTODO PROPOSTO

Neste capítulo são apresentadas as adequações do Algoritmo Genético proposto para a resolução do FSP e CFSP. A primeira seção descreve os experimentos realizados a priori para configurar o algoritmo de resolução dos dois problemas, com destaque para a apresentação do operador mutação.

5.1 ALGORITMO GENÉTICO APLICADO AO FSP E CFSP

Como já explicitado nesse trabalho, o objetivo do AG proposto consiste em determinar uma sequência, para o FSP e CFSP, que processe todas as tarefas nas máquinas em um tempo mínimo atendendo aos requisitos de cada problema. Nos Capítulos III e IV, vimos como obter a função de avaliação de cada problema. A seguir definem-se as componentes usadas na construção do nosso algoritmo genético, seguindo a estrutura das oito componentes de um AG.

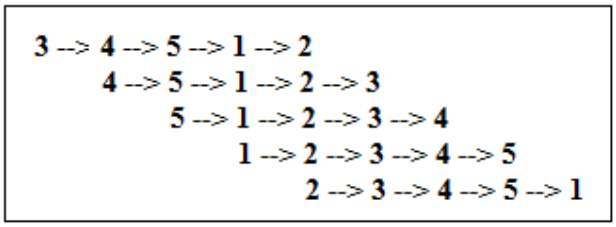
a) O cromossomo representa uma permutação das *n* tarefas, definindo uma sequência de produção ou uma solução viável do problema.

b) A função custo (*The fitness function*) do FSP é dada pelo procedimento *g* no Capítulo III, Figura 3. Já, para o CFSP, ela é dada pelas equações 4.1 e 4.2, no Capítulo IV. Ambas tratam do *makespan* como critério de desempenho dos dois problemas.

c) O AG proposto tem uma população inicial (*NPop*) com 200 indivíduos, selecionados pela heurística *PopInic*, descrita na Figura 14. As 200 melhores soluções destas heurísticas serão inseridas na população inicial do AG obedecendo a ordem crescente no valor na função custo (*makespan*). Desta forma, os melhores indivíduos da população são P_1, P_2, ..., P_{200}, nesta ordem. A heurística *PopInic*, desenvolvida por nós, produz $n\times(n\text{-}1)$ soluções viáveis do problema. Ela é equivalente a *heurística do vizinho mais próximo* bastante aplicada no Problema do Caixeiro Viajante, onde se o melhor vizinho da tarefa *j* é a tarefa *k*, então o processamento das tarefas *j* e *k*, consequentemente, deixa a menor ociosidade na

última máquina (M_m). A heurística produz *n* soluções usando este procedimento, iniciando por cada uma das *n* tarefas. Para cada uma destas soluções tem-se a geração de outras (*n*-1) soluções a partir dela, da seguinte forma: sejam *n*=5 e *s*=<3 4 5 1 2>, uma solução encontrada pelo procedimento do vizinho mais próximo, então serão geradas e avaliadas as soluções <4 5 1 2 3>, <5 1 2 3 4>, <1 2 3 4 5> e <2 3 4 5 1>. Cada uma das *n*-1 novas soluções muda a ordem de processamento da primeira tarefa seguindo a ordem da solução inicial encontrada, conforme ilustra a Figura 14 a seguir.

Figura 14 – Ilustração da aplicação da heurística *PopInic*.



Fonte: Elaborado pelo Autor.

d) A Seleção para cruzamento e mutação será feita com todos os indivíduos da população.

e) O cruzamento será feito entre todos os indivíduos da população, dando 19.900 cruzamentos (combinação de 200 dois a dois).

O operador cruzamento (*crossover*) é definido da seguinte forma. Primeiro divida a solução (permutação) em 3 partes, sendo dois pontos de corte c_1 e c_2, com $1 \leq c_1 \leq n$, $1 \leq c_2 \leq n$ e $c_1 \neq c_2$. O operador *crossover* implementado foi o *Order Crossover* de dois pontos (2P) (Goldberg, 1989). O operador *crossover* 2P foi criado baseado na ideia dos bons blocos construídos. Por isso, baseia-se nas posições relativa e absoluta dos seus genes. Este procedimento é descrito a seguir:

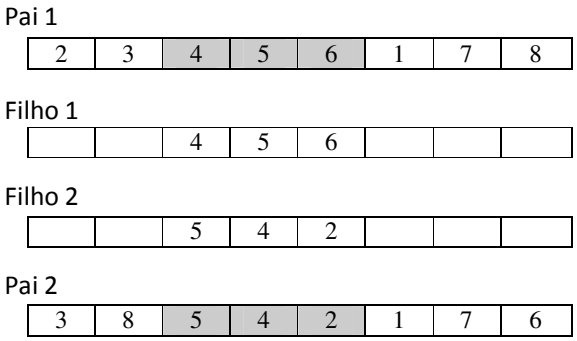
1 – Dois pais são escolhidos através do método de seleção (Pai 1 e 2);

2 – Com base nos 2 pontos de cortes determinados, o cromossomo foi dividido em três blocos, conforme mostra a Figuras 5.2. O segundo bloco de cada cromossomo é copiado para cada um dos Filhos (O_1 e O_2). Esta etapa preserva as posições absoluta e relativa das

estruturas do cromossomo de cada pai em cada filho, dando a ideia de que haja hereditariedade; e

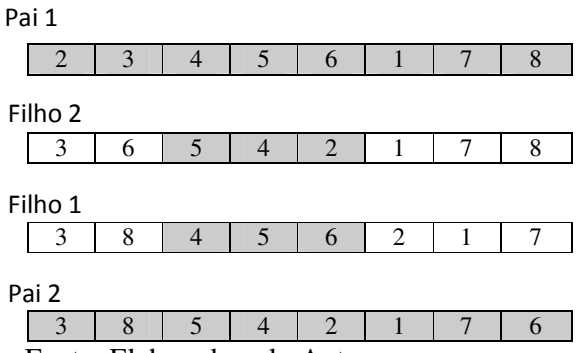
3 – As posições não-preenchidas de cada filho são copiadas das posições do outro pai no sentido da esquerda para a direita (Figura 16). Este procedimento faz com que seja preservada a ordem relativa das tarefas nas sequencias.

Figura 15 – Segunda etapa do *crossover* 2P.



Fonte: Elaborado pelo Autor.

Figura 16 – Terceira etapa do *crossover* 2P.



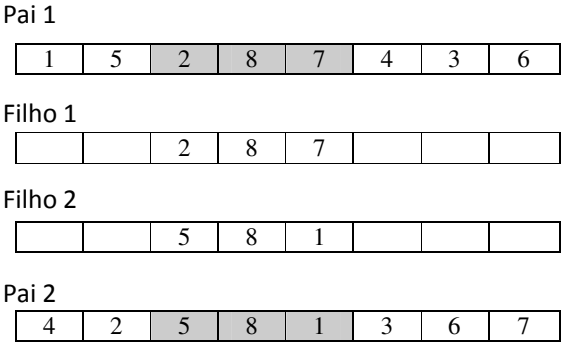
Fonte: Elaborado pelo Autor.

Neste exemplo o valor de n=8, c_1=3 e c_2=5, onde: $c_1=\lfloor n/3 \rfloor+1$e $c_2=2\times\lfloor n/3 \rfloor + 1$.

Também usamos outras combinações de blocos no cruzamento acima. Testamos o *crossover* de um ponto (1P) onde c_1=n/2. Um método mais moderno de *crossover* denominado *Partially Matched* (PM) *Crossover* também foi implementado e analisado. Segue uma ilustração deste método na Figura 17 e 5.5 abaixo. Ele também usa dois pontos de corte, que podem ser c_1 e c_2 dados anteriormente. O segundo bloco do Pai 1 é copiado para o Filho

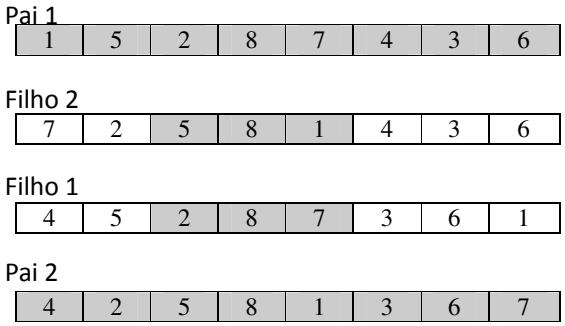
1, assim como o primeiro e terceiro bloco do Pai 2. Depois serão feitos os ajustes para a geração de uma solução viável, representada pelo Filho 1, substituindo no primeiro e terceiro bloco os genes repetidos do Filho 1, pelos genes que se encontram na mesma posição daqueles repetidos que estão no segundo bloco do Pai 2. Estes genes são aqueles que não se encontram no Filho 1 após as cópias dos blocos, no caso específico do exemplo dado através das Figuras 5.4 e 5.5, são as tarefas 5 e 1. O procedimento também se aplica da mesma forma para o Pai 2 e Filho 2.

Figura 17 – Segunda etapa do *PM crossover*.



Fonte: Elaborado pelo Autor.

Figura 18 – Terceira etapa do *PM crossover*.

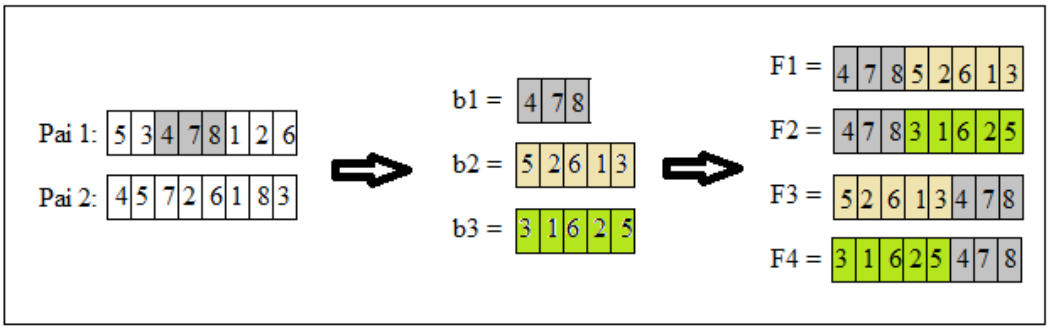


Fonte: Elaborado pelo Autor.

Desenvolveu-se um *crossover* denominado de 2-Blocos (2B), para aumentar o número de soluções geradas pelo AG, com intuito de diversificar mais ainda a busca pela solução ótima do problema. A Figura 19, dada a seguir, ilustra este procedimento. Cada uma das soluções Pai 1 e Pai 2 são divididas em três blocos. Tomando o Pai 1 como sendo a base, os blocos b_1 e b_2 vão gerar quatro filhos: $F_1=b_1b_2$, $F_2= b_1b_3$, $F_3= b_2b_1$ e $F_4= b_3b_1$. O bloco b_1 é

igual ao bloco do meio do Pai 1. O bloco b_2 é composto pelos elementos do Pai 2 que não estão em b_1, ordenados da esquerda para a direita. O bloco b_3 tem a ordem inversa dos elementos do bloco b_2. O mesmo procedimento é repetido para o Pai 2 sendo a base, gerando mais 4 filhos. Os pontos de corte continuam sendo c_1 e c_2, dados anteriormente, para gerar os três blocos.

Figura 19 – Ilustração do *crossover* 2B.



Fonte: Elaborado pelo Autor.

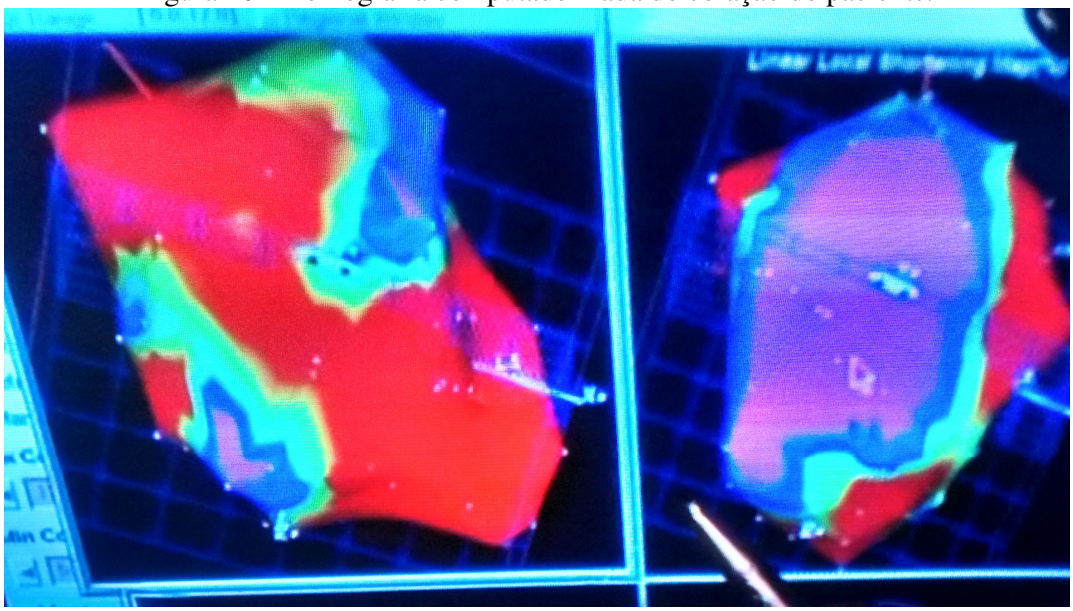
O Operador Mutação a ser utilizado neste AG é baseado numa técnica bem sucedida na regeneração dos músculos cardíacos através de aplicação de células-tronco que adaptamos para ser empregada na resolução de problema de otimização combinatória. As células-tronco, células-mães ou células estaminais são células que possuem a melhor capacidade de se dividir dando origem a duas células semelhantes às progenitoras.

As células-tronco de embriões têm ainda a capacidade de se transformar, num processo também conhecido por diferenciação celular, em outros tecidos do corpo, como ossos, nervos, músculos e sangue. Devido a essa característica, as células-tronco são importantes, principalmente na aplicação terapêutica, sendo potencialmente úteis em terapias de combate a doenças cardiovasculares, neurodegenerativas, diabetes mellitus tipo 1, acidentes vasculares cerebrais, doenças hematológicas, traumas na medula espinhal e nefropatias. O principal objetivo das pesquisas com células-tronco é usá-las para recuperar tecidos danificados por essas doenças e traumas.

Pesquisas realizadas pelos cientistas James Willerson e Emerson Perin, do Instituto do Coração do Texas-USA, mostram a eficiência da aplicação de células-tronco na

regeneração de pacientes com problemas cardiovasculares. Existem diversos artigos na literatura que tratam da terapia de células-tronco tais como HOCHEDLINGER e JAENISH (2003), Perin e Willerson (2011) e Perin e Willerson (2012). A Figura 20 mostra as imagens, via tomografias computadorizadas, de como estava e ficou o coração de um paciente, que foi tratado com aplicação de células-tronco. A imagem do lado esquerdo ilustra o coração do paciente, antes da aplicação de células-tronco, onde os pontos vermelhos indicam que as células do músculo deste coração estão bastante danificadas. A imagem do lado direito mostra como as células do músculo do coração do paciente se regeneraram depois da aplicação de células-tronco (saindo da cor vermelha para a cor branca) e consequentemente a melhora do coração do paciente.

Figura 20 – Tomografia computadorizada do coração do paciente.



Fonte: Elaborado pelo Autor.

Esta técnica funciona da seguinte forma, células-tronco são retiradas da medula osséa do paciente e tratadas em laboratório. As células-tronco são analisadas e aquelas com características fortes são aplicadas na região muscular do coração do paciente que está degenerada. Ainda é um mistério para a medicina como se dá esta regeneração. A lógica deste modelo é que as células-tronco entram em contato com as células ruins do músculo do coração e após algum tempo é feita a regeneração dessas células. Este processo é multiplicador fazendo com que grande parte das células vizinhas também tenha o mesmo

tratamento, sendo que em alguns casos nas regiões circunvizinhas, onde foi aplicado a células-tronco, não se obteve tanto sucesso quanto em outras regiões.

Adaptamos esta técnica para ser aplicada na resolução de problemas de otimização combinatória permutacional, cuja melhor solução encontrada até o momento, num determinado instante de execução do método de resolução do problema, fará o papel da célula-tronco. Esta, por sua vez, vai poder modificar ou não as demais soluções da população atual, com o intuito de regenerá-las ou não. O processo de regeneração dar-se-á levando em consideração que uma ou mais cópias de partes distintas da célula-tronco será copiada na solução que se quer regenerá-la. A cópia dar-se-á da seguinte forma:

1º) Vê-se quem é a tarefa na 1ª posição da solução a ser regenerada e faz j=1;

2º) Seja h a posição desta tarefa na solução considerada célula-tronco. Faz-se uma cópia das k tarefas seguintes a h, na mesma ordem em que se encontram na célula-tronco, e coloca-a na solução a ser regenerada a partir da (j+1)-ésima posição. Caso haja menos do que k tarefas a partir de h, i. e. $h+k>n$, copia-se apenas as posições disponíveis ($k=n-j$), ou não será feita cópia, se todas as tarefas a direita de h já tenham sido copiadas;

3º) Atualiza-se $j \leftarrow j+1$, verifica-se quem é a tarefa que está localizada na j-ésima posição da solução a ser regenerada. Caso esta tarefa já tenha sido copiada anteriormente na solução regenerada, passa-se para a tarefa seguinte até encontrar uma tarefa que ainda não tenha sido copiada anteriormente e faz-se o procedimento executado no 2º passo;

4º) Este processo termina quando $j=n$.

A Figura 21, dada a seguir, ilustra a aplicação deste procedimento, através de um exemplo com n=8 e k=2, com seu código em linguagem C. A solução a ser regenerada vai herdar as boas características da célula-tronco. Outra característica importante desta técnica é que a célula-tronco a ser usada é atualizada sempre que uma solução melhor for encontrada.

Figura 21 – Ilustração do procedimento aplicação de célula-tronco.

```
Procedimento células-tronco:
Input: n, k, sopt, s;
Output: sr (solução s regenerada);

   for (i=1; i<=n; i++) { set[i]=0; sr[i]=0; }
   k1=0;
   for (i=1; i<=n; i++) {
     if (set[ s[i] ]==0) {
        j=1; k1=k1+1;
        while (j<=n) {
          if (s[i]==sopt[j]) { a=j; j=n; }
          j=j+1;
        } // end while
        sr[k1]=s[i];
        set[s[i]]=set[s[i]]+1;
        j=a+1; k2=1;
        while ((j<=n) && (k2<=k)) {
          if (set[ sopt[j] ]==0) {
            k1=k1+1;
            sr[k1]=sopt[j];
            k2=k2+1;
            set[ sopt[j] ]= set[ sopt[j] ] +1;
          }
          j=j+1;
        }
     } //end if
   } // end for
```

Célula-tronco:

3	5	8	6	1	7	2	4

Soluçao da População:

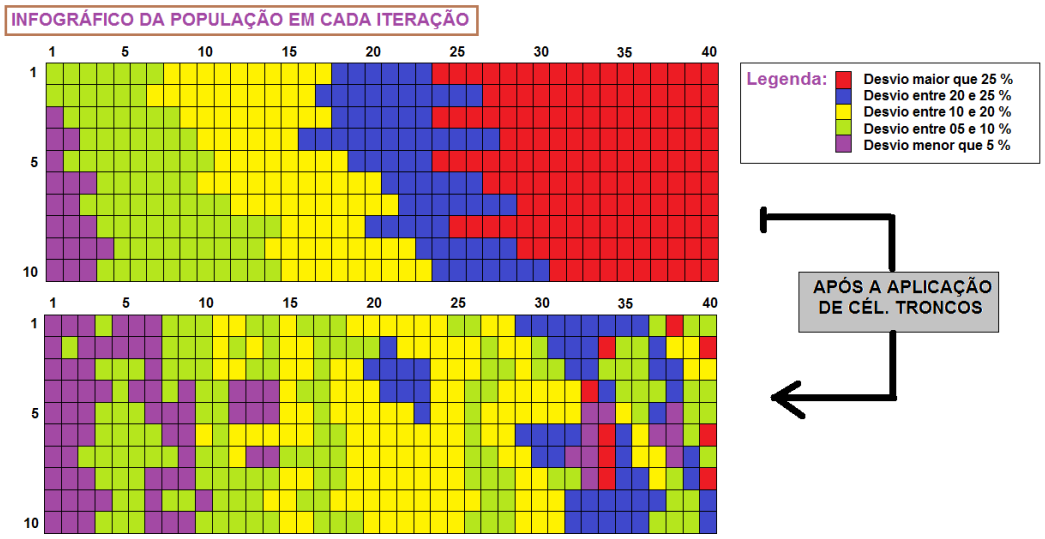
7	3	1	5	2	6	8	4

Solução Regenerada:

7	2	4	3	5	8	1	6

Fonte: Elaborado pelo Autor.

Figura 22 – Infográfico de uma aplicação do AG com 10 iterações.



Fonte: Elaborado pelo Autor.

A Figura 22, dada acima, ilustra através de um infográfico, o desempenho pretendido pelo operador mutação aplicado a um problema de otimização, equiparando-se ao procedimento

analítico, no caso do paciente com o coração bastante danificado, através da tomografia computadorizada do seu coração. Será possível verificar este desempenho de forma bastante rápida, devido as cores associadas aos desvios das soluções da população, em cada iteração, indicando o percentual de aproveitamento de cada solução da população antes e depois da regeneração.

f) A estratégia geracional é responsável por controlar a substituição de indivíduos de uma geração para a outra na população. A estratégia geracional proposta para o nosso AG é substituir integralmente todos os indivíduos da população pelos melhores indivíduos encontrados no operador mutação e cruzamento, com isso pretende-se diversificar e intensificar mais ainda a busca pela solução ótima do problema.

g) Inicialmente o critério de parada usava o limite de valores propostos para três variáveis: número de iterações do algoritmo, tempo máximo de execução e a média dos *fitness* das soluções da população. Depois das análises feitas com os testes de execução do algoritmo com vários problemas da literatura, verificou-se que nunca o algoritmo parou para os valores das duas primeiras variáveis: número de iterações do algoritmo no máximo igual a *n* e tempo máximo de execução limitado a uma hora. Para a terceira variável foi usado o valor médio na função objetivo das 200 soluções da população atual e comparado com o valor médio da população imediatamente anterior, quando estes valores forem iguais, em pelo menos 3 iterações seguidas o algoritmo para e apresenta a melhor solução encontrada como solução do problema. Com isto evitamos avaliações de soluções que podem ser distintas, mas que tiveram o mesmo desempenho de soluções similares, muitas vezes estas soluções são repetidas, foi o que se constatou nos experimentos. Isso evitou o desgaste no tempo de execução. Porém, com este critério, o AG estava convergindo muito rápido e resolveu-se adotar uma mudança radical nos elementos da última população atual. Iniciar a busca com todos os elementos da população sendo alterados da seguinte forma:

```
for (i=1; i<=NPop; i++)
   for (j=1; j<=n; j++) p_ij = n + 1 – p_ij .
```

Desta forma, se uma solução da população fosse s=< 4 3 2 6 8 7 5 1 >, após a aplicação desse critério, ela ficaria < 5 6 7 3 1 2 4 8 >, ou seja, cada elemento de s seria trocado pelo seu

simétrico em relação a *n*. Isto criou uma oportunidade para AG visitar regiões não atingidas anteriormente no espaço de busca.

h) A última etapa de um AG é a calibração dos valores dos seus parâmetros, como tamanho da população, taxa de *crossover*, entre outras. Segundo Mitchell (1998) os parâmetros de um AG interagem entre si de forma não-linear sendo de difícil calibração. Muitos trabalhos têm sido realizados nesta área sem opinião consensual (Ruiz *et al.*, 2006) e uma desvantagem do AG é a dificuldade do processo de calibração dos seus parâmetros (Dréo *et al.*, 2006). Vários testes foram realizados para determinar os valores de *NPop* e qual ou quais o(s) operador(es) *crossover* que deve(m) ser utilizado(s) (1P, 2P, PM e 2B).

Tabela 4 – Desempenho do AG para determinar o valor de *NPop*.

Instância	RAJ	180	190	200	210	220	230	240	250	Mínimo	Desvio	GASA	TS-M
rec01	1590	**1526**	**1526**	**1526**	**1526**	**1526**	**1526**	**1526**	**1526**	1526	-4,03	-3,96	-3,96
rec03	1457	**1361**	**1361**	**1361**	**1361**	**1361**	**1361**	**1361**	**1361**	1361	-6,59	-4,46	-6,59
rec05	1637	**1511**	**1511**	**1511**	**1511**	**1511**	**1511**	**1511**	**1511**	1511	-7,70	-6,90	-7,64
rec07	2119	**2042**	**2042**	**2042**	**2042**	**2042**	**2042**	**2042**	**2042**	2042	-3,63	-3,45	-3,63
rec09	2141	2043	2043	**2042**	**2042**	**2042**	**2042**	**2042**	**2042**	2042	-4,62	-4,48	-4,58
rec11	1946	**1881**	**1881**	**1881**	**1881**	**1881**	**1881**	**1881**	**1881**	1881	-3,34	-3,34	-3,34
hel2	189	**179**	**179**	**179**	**179**	**179**	**179**	**179**	**179**	179	-5,29	-4,76	-5,29
rec13	2709	**2545**	**2545**	**2545**	**2545**	**2545**	**2545**	**2545**	**2545**	2545	-6,05	-5,65	-6,05
rec15	2691	**2529**	**2529**	**2529**	**2529**	**2529**	**2529**	**2529**	**2529**	2529	-6,02	-6,02	-6,02
rec17	2740	**2587**	**2587**	2603	2588	2588	2588	**2587**	**2587**	2587	-5,58	-5,47	-5,58
rec19	3157	2865	2854	2854	2855	2855	**2850**	2892	2882	2850	-9,72	-5,45	-9,25
rec21	3015	2832	2827	2829	2829	**2822**	2837	2837	**2822**	2822	-6,40	-2,22	-6,30
rec23	3030	2723	2703	2707	2716	**2700**	2726	2705	2723	2700	-10,89	-6,70	-10,73
rec25	3835	3596	**3593**	**3593**	3596	3600	3600	3600	3596	3593	-6,31	-2,69	-6,31
rec27	3655	**3431**	3439	**3431**	3444	**3431**	**3431**	3432	3432	3431	-6,13	-2,60	-6,10
rec29	3583	3315	3394	**3291**	3359	3312	3352	3333	3325	3291	-8,15	-3,99	-8,28
rec31	4631	4382	4415	4359	4411	**4339**	4391	4390	4425	4339	-6,31	2,72	-6,13
rec33	4770	4469	4457	4472	**4433**	4472	4543	4488	4488	4433	-7,06	4,78	-6,31
rec35	4718	4465	4446	4447	4453	4486	4479	4473	**4405**	4405	-6,63	3,67	-6,17
rec37	8979	8269	8178	8273	8130	8186	**8097**	8116	8115	8097	-9,82	5,89	-9,49
rec39	9158	**8672**	8868	8772	9355	8793	8678	8721	8750	8672	-5,31	8,80	-6,99
rec41	9344	9086	8700	**8643**	8707	8750	8913	8702	8774	8643	-7,50	6,79	-8,57
hel1	780	727	732	722	722	**720**	725	727	726	720	-7,69	12,44	-8,21
Desvio Médio		-6,00	-6,05	-6,27	-5,92	-6,25	-6,04	-6,15	-6,16		-6,56	-1,18	-6,59

Fonte: Elaborado pelo Autor.

Os resultados dos experimentos realizados nas instâncias do CFSP, descritos aqui, foram para determinar estes dois parâmetros especificamente. As Tabelas dadas mostram o desempenho do AG nas 23 instâncias de Reeves (1995) e Heller (1960). A Tabela 4, dada anteriormente, mostra o desempenho de AG para o tamanho da população *NPop* variando de 180 a 250 indivíduos. Foram realizados testes para valores menores, com $30 \leq Npop \leq 170$, onde se constatou um fraco desempenho. As colunas da Tabela 4 representam: a instância do problema atacado; o valor da solução encontrada pelo algoritmo RAJ; o valor encontrado por AG com *NPop* variando de 180 a 250; o menor valor encontrado por AG dentro da variação dada para *NPop*; o desvio dado por $100\times(z - z^*)/z^*$, onde z é o valor de g na melhor solução encontrada pelo método de resolução do problema e z^* é o valor encontrado pelo algoritmo de RAJ; o desvio do algoritmo GASA; e o desvio do algoritmo TS-M, considerado o melhor algoritmo de resolução do problema.

A Tabela 4 mostra que o melhor desempenho do AG aconteceu para *NPop* igual a 200, com desvio médio de -6,27%, melhor que o desempenho do GASA e um pouco acima (0,32%) do TS-M, que foi de -6,59. Os valores em negritos determinam onde foram encontrados os melhores desempenhos do AG. O AG com *NPop* igual a 200 teve 13 desempenhos igual ao *mínimo*, enquanto o AG com *NPop* igual a 220 teve 14 desempenhos igual ao *mínimo*, porém com desvio médio pior que o de *NPop* igual a 200. Se o AG fosse executado oito vezes, uma para cada variação de *NPop*=180, 190, 200, ..., 250, o seu desempenho (-6,56%) ficaria quase igual ao TS-M. Este AG foi executado com todos os quatro tipos de cruzamentos, dado pela seguinte ordem 1P, 2P, PM e 2B. Resolveu-se adotar para o AG aqui proposto *NPop* sendo igual a 200.

As Tabelas 5 a 8, dadas a seguir, mostram os resultados do AG quando o objeto de análise é verificar qual o tipo de *crossover* deve ser usado, simples ou combinados. As Tabelas 5 a 8 apresentam em suas colunas: a instância do problema atacado; os valores de m e n para cada instância do problema; o valor da solução encontrada pelo algoritmo RAJ; o melhor valor encontrado por AG para cada um dos operadores *População Inicial* (Pop. In.), *Cruzamento* (Cruz.) e *Mutação* (Mut.); o menor valor encontrado por AG dentro da variação de cada operador genético; o desvio dado igual aquele usado na Tabela 4; e o tempo de execução do AG dado em segundos.

Tabela 5 – Resultados do AG com *crossover* 1P.

Instância	n x m	RAJ	Pop. In.	Cruz.	Mut.	Mínimo	Desvio	Tempo (s)
rec01	20x5	1590	2021	1526	1526	1526	-4,03	1,53
rec03	20x5	1457	1904	1389	1389	1389	-4,67	1,56
rec05	20x5	1637	1864	1524	1524	1524	-6,90	1,39
rec07	20x10	2119	2596	2053	2053	2053	-3,11	1,58
rec09	20x10	2141	2610	2075	2075	2075	-3,08	1,30
rec11	20x10	1946	2385	1881	1881	1881	-3,34	1,47
hel2	20x10	189	228	182	182	182	-3,70	1,61
rec13	20x15	2709	3231	2553	2553	2553	-5,76	1,48
rec15	20x15	2691	3384	2529	2529	2529	-6,02	1,27
rec17	20x15	2740	3206	2607	2607	2607	-4,85	1,34
rec19	30x10	3157	3918	3009	3009	3009	-4,69	1,75
rec21	30x10	3015	3807	2897	**2894**	2894	-4,01	2,06
rec23	30x10	3030	3897	2782	2782	2782	-8,18	2,42
rec25	30x15	3835	4907	3706	3706	3706	-3,36	1,91
rec27	30x15	3655	4779	3579	**3577**	3577	-2,13	2,36
rec29	30x15	3583	4986	3497	3497	3497	-2,40	2,42
rec31	50x10	4631	6383	4970	**4943**	4943	6,74	3,13
rec33	50x10	4770	6587	4658	4658	4658	-2,35	4,36
rec35	50x10	4718	6930	4696	4696	4696	-0,47	2,92
rec37	75x20	8979	12889	8995	8995	8995	0,18	5,39
rec39	75x20	9158	12841	9084	9084	9084	-0,81	6,42
rec41	75x20	9344	13219	9316	**9315**	9315	-0,31	5,55
hel1	100x10	780	1114	786	**785**	785	0,64	8,83
			Média				**-2,90**	**2,78**
			Mínimo				**-8,18**	**1,27**
			Máximo				**6,74**	**8,83**

Fonte: Elaborado pelo Autor.

Tabela 6 – Resultados do AG com *crossover* 2P.

Instância	n x m	RAJ	Pop. In.	Cruz.	Mut.	Mínimo	Desvio	Tempo (s)
rec01	20x5	1590	2021	1526	1526	1526	-4,03	4,95
rec03	20x5	1457	1904	1375	1375	1375	-5,63	3,86
rec05	20x5	1637	1864	1512	1512	1512	-7,64	4,39
rec07	20x10	2119	2596	2047	2047	2047	-3,40	3,98
rec09	20x10	2141	2610	2042	2042	2042	-4,62	4,38
rec11	20x10	1946	2385	1881	1881	1881	-3,34	4,55
hel2	20x10	189	228	179	179	179	-5,29	4,70
rec13	20x15	2709	3231	2545	2545	2545	-6,05	4,25
rec15	20x15	2691	3384	2529	2529	2529	-6,02	3,86
rec17	20x15	2740	3206	2607	2607	2607	-4,85	3,92
rec19	30x10	3157	3918	2936	2936	2936	-7,00	5,33
rec21	30x10	3015	3807	2844	2844	2844	-5,67	7,00
rec23	30x10	3030	3897	2758	2758	2758	-8,98	5,55
rec25	30x15	3835	4907	3604	3604	3604	-6,02	5,91
rec27	30x15	3655	4779	3461	3461	3461	-5,31	5,17
rec29	30x15	3583	4986	3385	3385	3385	-5,53	5,66
rec31	50x10	4631	6383	4389	4389	4389	-5,23	13,91
rec33	50x10	4770	6587	4604	4604	4604	-3,48	13,98
rec35	50x10	4718	6930	4492	4492	4492	-4,79	14,16
rec37	75x20	8979	12889	8429	8429	8429	-6,13	15,88
rec39	75x20	9158	12841	9243	9243	9243	0,93	15,44
rec41	75x20	9344	13219	9207	9207	9207	-1,47	17,70
hel1	100x10	780	1114	768	768	768	-1,54	25,30
			Média				**-4,83**	**8,43**
			Mínimo				**-8,98**	**3,86**
			Máximo				**0,93**	**25,30**

Fonte: Elaborado pelo Autor.

Tabela 7 – Resultados do AG com *crossover* PM.

Instância	n x m	RAJ	Pop. In.	Cruz.	Mut.	Mínimo	Desvio	Tempo (s)
rec01	20x5	1590	2021	1537	1537	1537	-3,33	1,05
rec03	20x5	1457	1904	1375	1375	1375	-5,63	0,95
rec05	20x5	1637	1864	1511	1511	1511	-7,70	1,05
rec07	20x10	2119	2596	2047	2047	2047	-3,40	1,11
rec09	20x10	2141	2610	2045	2045	2045	-4,48	0,86
rec11	20x10	1946	2385	1940	1940	1940	-0,31	0,94
hel2	20x10	189	228	185	185	185	-2,12	0,88
rec13	20x15	2709	3231	2641	**2618**	2618	-3,36	0,95
rec15	20x15	2691	3384	2557	2557	2557	-4,98	0,80
rec17	20x15	2740	3206	2674	2674	2674	-2,41	0,77
rec19	30x10	3157	3918	3001	3001	3001	-4,94	1,33
rec21	30x10	3015	3807	2872	2872	2872	-4,74	1,78
rec23	30x10	3030	3897	2787	2787	2787	-8,02	1,44
rec25	30x15	3835	4907	3721	**3710**	3710	-3,26	1,97
rec27	30x15	3655	4779	3513	3513	3513	-3,89	1,34
rec29	30x15	3583	4986	3444	3444	3444	-3,88	1,58
rec31	50x10	4631	6383	4632	**4546**	4546	-1,84	2,33
rec33	50x10	4770	6587	4615	**4597**	4597	-3,63	3,41
rec35	50x10	4718	6930	4638	4638	4638	-1,70	2,23
rec37	75x20	8979	12889	9251	9251	9251	3,03	4,41
rec39	75x20	9158	12841	9312	**9265**	9265	1,17	6,72
rec41	75x20	9344	13219	9680	9680	9680	3,60	4,00
hel1	100x10	780	1114	760	760	760	-2,56	11,61
			Média				**-2,97**	**2,33**
			Mínimo				**-8,02**	**0,77**
			Máximo				**3,60**	**11,61**

Fonte: Elaborado pelo Autor.

Os resultados das Tabelas 5 a 8 mostram, em negritos, as instâncias em que o operador mutação teve melhor desempenho que os demais operadores do AG. Isto comprova a eficiência e eficácia deste operador, encontrando boas soluções, fazendo dele um operador que intensifica a busca pela solução ótima do problema. Desta forma, o processo usado no experimento genético também está sendo verificado no procedimento similar aqui adotado, as

células-tronco (boas soluções do problema) estão regenerando parte das células degeneradas (soluções de baixo desempenho).

Tabela 8 – Resultados do AG com *crossover* 2B.

Instância	n x m	RAJ	Pop. In.	Cruz.	Mut.	Mínimo	Desvio	Tempo (s)
rec01	20x5	1590	2021	1527	1527	1527	-3,96	3,16
rec03	20x5	1457	1904	1399	**1390**	1390	-4,60	3,38
rec05	20x5	1637	1864	1511	1511	1511	-7,70	4,41
rec07	20x10	2119	2596	2046	2046	2046	-3,45	3,20
rec09	20x10	2141	2610	2056	**2042**	2042	-4,62	2,89
rec11	20x10	1946	2385	1890	1890	1890	-2,88	3,67
hel2	20x10	189	228	180	180	180	-4,76	2,39
rec13	20x15	2709	3231	2623	**2552**	2552	-5,80	2,73
rec15	20x15	2691	3384	2549	**2530**	2530	-5,98	3,11
rec17	20x15	2740	3206	2603	2603	2603	-5,00	3,33
rec19	30x10	3157	3918	2939	**2928**	2928	-7,25	6,64
rec21	30x10	3015	3807	2880	2880	2880	-4,48	5,20
rec23	30x10	3030	3897	2711	2711	2711	-10,53	5,86
rec25	30x15	3835	4907	3609	3609	3609	-5,89	6,75
rec27	30x15	3655	4779	3490	3490	3490	-4,51	5,11
rec29	30x15	3583	4986	3394	**3375**	3375	-5,81	5,06
rec31	50x10	4631	6383	4438	**4432**	4432	-4,30	8,73
rec33	50x10	4770	6587	4623	4623	4623	-3,08	10,11
rec35	50x10	4718	6930	4536	4536	4536	-3,86	13,72
rec37	75x20	8979	12889	8538	**8530**	8530	-5,00	15,25
rec39	75x20	9158	12841	9189	**9181**	9181	0,25	11,81
rec41	75x20	9344	13219	8965	8965	8965	-4,06	17,66
hel1	100x10	780	1114	764	**763**	763	-2,18	17,41
			Média				**-4,76**	**7,03**
			Mínimo				**-10,53**	**2,39**
			Máximo				**0,25**	**17,66**

Fonte: Elaborado pelo Autor.

A Tabela 9, dada a seguir, mostra um resumo das informações existentes nas Tabelas 5 a 8, onde são destacados: os desvios e o tempo de execução de cada operador crossover para cada instância; a média, o mínimo e o máximo dos desvios para cada operador; e o número de problemas que o operador teve melhor desempenho.

Tabela 9 – Resultados tabulados dos dados das Tabelas 5 a 8.

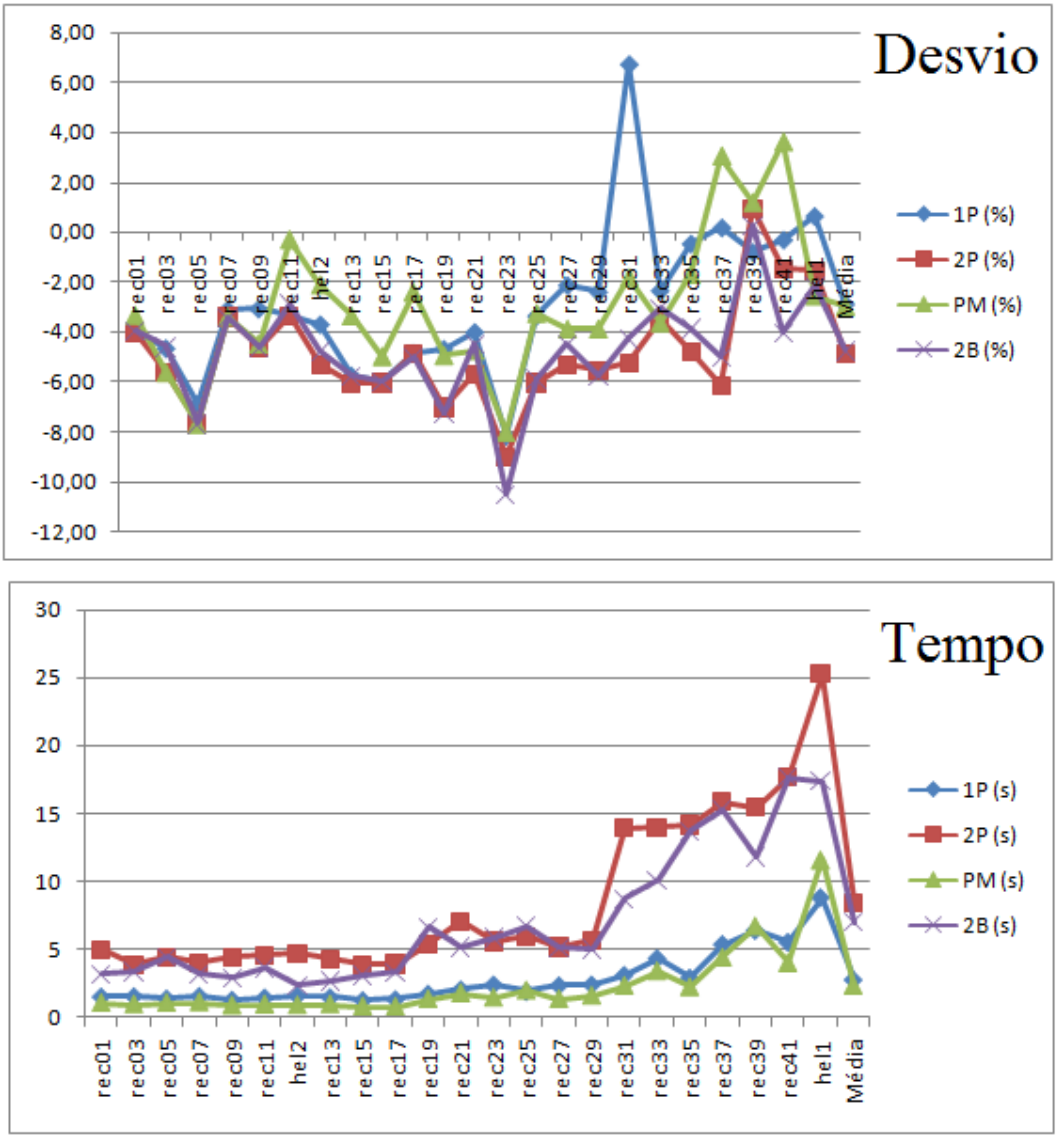
Instância	n x m	1P (%)	2P (%)	PM (%)	2B (%)	1P (s)	2P (s)	PM (s)	2B (s)
rec01	20x5	**-4,03**	**-4,03**	-3,33	-3,96	1,53	4,95	1,05	3,16
rec03	20x5	-4,67	**-5,63**	**-5,63**	-4,60	1,56	3,86	0,95	3,38
rec05	20x5	-6,90	-7,64	**-7,70**	**-7,70**	1,39	4,39	1,05	4,41
rec07	20x10	-3,11	-3,40	-3,40	**-3,45**	1,58	3,98	1,11	3,2
rec09	20x10	-3,08	**-4,62**	-4,48	**-4,62**	1,30	4,38	0,86	2,89
rec11	20x10	**-3,34**	**-3,34**	-0,31	-2,88	1,47	4,55	0,94	3,67
hel2	20x10	-3,70	**-5,29**	-2,12	-4,76	1,61	4,70	0,88	2,39
rec13	20x15	-5,76	**-6,05**	-3,36	-5,80	1,48	4,25	0,95	2,73
rec15	20x15	**-6,02**	**-6,02**	-4,98	-5,98	1,27	3,86	0,80	3,11
rec17	20x15	-4,85	-4,85	-2,41	**-5,00**	1,34	3,92	0,77	3,33
rec19	30x10	-4,69	-7,00	-4,94	**-7,25**	1,75	5,33	1,33	6,64
rec21	30x10	-4,01	**-5,67**	-4,74	-4,48	2,06	7,00	1,78	5,2
rec23	30x10	-8,18	-8,98	-8,02	**-10,53**	2,42	5,55	1,44	5,86
rec25	30x15	-3,36	**-6,02**	-3,26	-5,89	1,91	5,91	1,97	6,75
rec27	30x15	-2,13	**-5,31**	-3,89	-4,51	2,36	5,17	1,34	5,11
rec29	30x15	-2,40	-5,53	-3,88	**-5,81**	2,42	5,66	1,58	5,06
rec31	50x10	6,74	**-5,23**	-1,84	-4,30	3,13	13,91	2,33	8,73
rec33	50x10	-2,35	-3,48	**-3,63**	-3,08	4,36	13,98	3,41	10,11
rec35	50x10	-0,47	**-4,79**	-1,70	-3,86	2,92	14,16	2,23	13,72
rec37	75x20	0,18	**-6,13**	3,03	-5,00	5,39	15,88	4,41	15,25
rec39	75x20	**-0,81**	0,93	1,17	0,25	6,42	15,44	6,72	11,81
rec41	75x20	-0,31	-1,47	3,60	**-4,06**	5,55	17,70	4,00	17,66
hel1	100x10	0,64	**-1,54**	-2,56	-2,18	8,83	25,30	11,61	17,41
Média		-2,90	**-4,83**	-2,97	-4,76	2,78	8,43	2,33	7,03
Mínimo		-8,18	-8,98	-8,02	**-10,53**	1,27	3,86	0,77	2,39
Máximo		6,74	0,93	3,60	**0,25**	8,83	25,30	11,61	17,66
Total		4	**14**	3	8				

Fonte: Elaborado pelo Autor.

A Figura 23 mostra o gráfico dos dados da Tabela 9, para o desvio e tempo de execução de cada operador, onde se constata o bom desempenho do operador 2P e 2B, individualmente. Estes dois operadores, 2P e 2B, obtiveram os melhores desempenhos com desvio médio de -4,83% e -4,76%, respectivamente. O operador 2P obteve o melhor desempenho em 14 das 23 instâncias testadas. O operador 2B obteve o melhor intervalo de desvio, com os menores mínimo (-10,53%) e máximo (0,25%). Os operadores 1P e PM

tiveram os menores tempos de execução do AG, quando comparados com 2P e 2B. Estes operadores (1P e PM) são bons para diversificar a busca, haja vista que eles não comprometem o tempo de execução do AG. Por esses motivos, aqui especificados, resolveu-se adotar o AG com os operadores *crossover* 1P, 2P, PM e 2B, nesta ordem de execução.

Figura 23 – Gráfico com o desvio e tempo de execução dos *crossovers*.



Fonte: Elaborado pelo Autor.

Portanto, o AG proposto para ser aplicado no FSP e CFSP tem o tamanho da população sendo igual a 200 indivíduos, a seleção é realizada com todos os indivíduos da população, os operadores crossover usados são 1P, 2P, PM e 2B (nesta ordem de execução), o operador mutação é o procedimento células-tronco e o critério de parada é aquele descrito no item (h) deste Capítulo. A seguir, serão descritos os resultados dos experimentos computacionais realizados com o AG tendo esta configuração.

6 EXPERIMENTOS COMPUTACIONAIS

Neste capítulo são apresentados os experimentos computacionais realizados com o AG proposto neste trabalho. A primeira seção é dedicada a execução da metaheurística Algoritmo Genético com aplicação de células-tronco no FSP, enquanto na segunda seção, a aplicação será no CFSP.

Vários experimentos computacionais foram realizados para observar o desempenho do Algoritmo Genético com células-tronco, denominado de AG. Os experimentos computacionais foram executados em um microcomputador PC Dell (3,2 Ghz e 4 Gb de RAM), tendo o código implementado em linguagem C/C++, do compilador Dev C++ na versão 4.9.9.2. O principal indicador utilizado entre os algoritmos de resolução do problema é o percentual de desvio das soluções, dado por $100\times(z - z^*)/z^*$, onde z é o valor de g na melhor solução encontrada pelo método de resolução do problema e z^* é o valor da melhor solução do problema. No caso do FSP z^* é o valor da solução ótima, enquanto que no CFSP, ele representa o valor encontrado pelo algoritmo de RAJ. Um conjunto de experimentos computacionais foi programado para ser realizado e os resultados são apresentados nas seções 6.1 e 6.2, dadas a seguir.

6.1 RESULTADOS DO AG APLICADO AO FSP

O AG foi aplicado nas 90 instâncias do FSP, através dos problemas consolidados na literatura, *benchmark* de Taillard (1993). As instâncias foram divididas em 9 Classes, com 10 problemas cada, conforme os valores de n (20, 50, 100) e m (5, 10, 20). As Classes estão definidas por: C1 (m=5), C2 (m=10) e C3 (m=20), com n=20; C4 (m=5), C5 (m=10) e C6 (m=20), com n=50; e C7 (m=5), C8 (m=10) e C9 (m=20), com n=100. Na Tabela 10, dada a seguir, encontram-se: a referência da classe testada; o percentual de desvio dos Algoritmos Genéticos de Chen (AGC), de Reeves (AGRe), de Murata (AGM) e de Ruiz (AGRu); e o percentual de desvio e o tempo de execução do AG proposto, com células-tronco. Não foi colocado o tempo de execução dos outros métodos devido eles terem sido executados em

máquinas distintas. O critério de parada de AGC, AGM, AGRe e AGRu levou em consideração o fator tempo dado por n×(m/2)×*p* milisegundos, onde *p* pode ser igual a 30, 60 ou 90.

Tabela 10 – Desempenho dos métodos para o FSP.

Método	p	C1	C2	C3	C4	C5	C6	C7	C8	C9	Geral
AGC	30	3,65	5,00	3,90	1,89	6,37	7,88	1,34	3,90	8,06	**4,67**
	60	4,02	5,14	3,93	2,02	6,83	7,98	1,44	3,78	8,18	**4,81**
	90	3,51	4,99	4,24	2,34	6,92	7,77	1,36	3,87	8,11	**4,79**
AGM	30	0,84	1,96	1,66	0,30	3,50	5,07	0,25	1,54	4,99	**2,23**
	60	0,74	1,72	1,66	0,26	3,20	4,88	0,25	1,46	4,77	**2,10**
	90	0,53	1,61	1,36	0,23	3,27	4,75	0,22	1,34	4,68	**2,00**
AGRe	30	0,54	1,78	1,39	0,17	2,23	3,74	0,14	0,82	3,36	**1,57**
	60	0,51	1,67	1,41	0,20	2,26	3,71	0,12	0,74	3,25	**1,54**
	90	0,62	1,71	1,31	0,16	2,00	3,58	0,11	0,67	3,12	**1,48**
AGRu	30	0,24	0,62	0,37	0,06	1,79	2,67	0,07	0,65	2,78	**1,03**
	60	0,23	0,60	0,34	0,06	1,86	2,62	0,08	0,62	2,68	**1,01**
	90	0,25	0,64	0,40	0,06	1,46	2,47	0,06	0,52	2,54	**0,93**
AG	**Média**	0,42	0,52	0,35	0,76	3,36	6,54	0,67	3,12	7,31	**2,56**
	Mínimo	0,00	0,00	0,00	0,07	1,45	5,13	0,04	1,98	4,95	**0,00**
	Máximo	1,49	1,69	0,67	1,31	5,85	9,02	2,11	4,35	9,17	**9,17**
	Tempo Médio	0,37	36,25	71,80	35,55	104,86	244,43	60,82	584,18	570,35	**189,85**

Fonte: Elaborado pelo Autor.

Analisando os dados da Tabela 10 têm-se os seguintes comentários:

a) O AGC, com *p*=30, 60 ou 90, foi o método com os piores desempenhos tendo média geral do percentual de desvio igual a 4,67%, 4,81% e 4,79, respectivamente. O AGRu, com p=30, 60 ou 90, foi o método com os melhores desempenhos tendo média geral do percentual de desvio igual a 1,03%, 1,01% e 0,93%, respectivamente. O AGRu é considerado o melhor método de resolução do FSP, depois dele vem o AGRe. Os métodos AGM e AG têm quase que o mesmo desempenho, sendo o AGM superior.

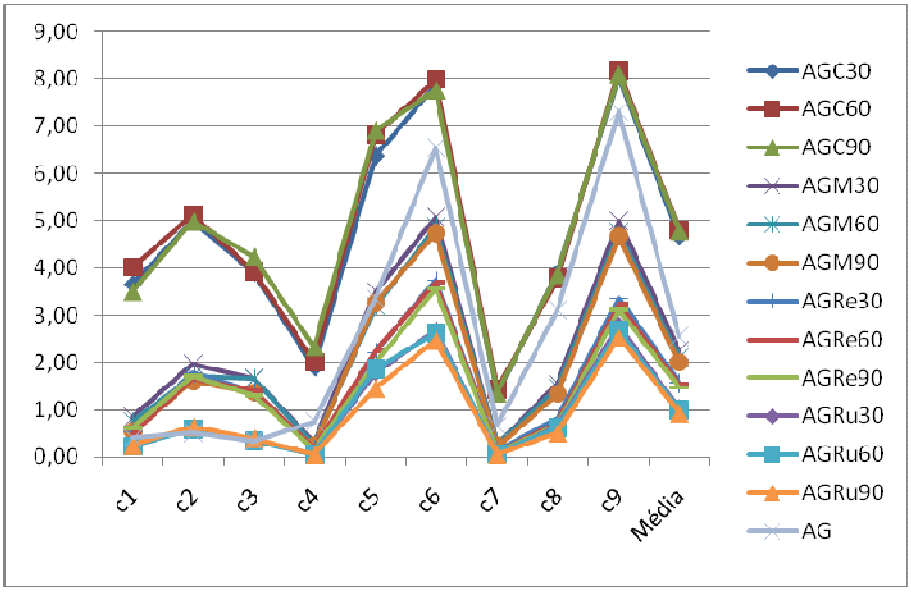
b) O método AG encontrou a solução ótima em 5 instâncias da classe C1, 3 instâncias da classe C2 e 1 instância da classe C3.

c) O AG foi o método que obteve o melhor desempenho nas classes 2 e 3, conforme mostra o gráfico da Figura 24, dada anteriormente.

d) O tempo médio de execução do AG variou de 0,37 segundos, na classe C1 (n=20, m=5), a 571 segundos, na classe C9 (n=100, m=20). Isto mostra como o AG proposto é rápido.

e) Todos os Algoritmos Genéticos usados na comparação com o nosso método são considerados Métodos Probabilísticos, enquanto que o nosso AG é considerado um Método Discreto, pois não usa lei de probabilidade para execução dos operadores, aleatoriedade na produção da população, aleatoriedade para a criação das soluções, entre outras. Não foi encontrado na literatura um AG discreto com um desempenho menor que 3,0 % para essas instâncias.

Figura 24 – Gráfico do percentual de desvio dos métodos de resolução do FSP.



Fonte: Elaborado pelo Autor.

6.2 RESULTADOS DO AG APLICADO AO CFSP

Este experimento tem como objetivo analisar o desempenho do AG no problema CFSP. Para isso o AG proposto foi testado com as instâncias de Reeves (1995) e Heller (1960). Os resultados do AG são comparados com o algoritmo híbrido (*Genetic Algorithm + Simulated Anealing*) de Shuster e Framinan (2003), denominado de GASA, único AG encontrado na literatura para o CFSP que utiliza instâncias conhecidas nos seus testes e o

melhor método de Grabowski e Pempera (2005) que é um Tabu Search com Multimovimento (TS-M) e cuja a solução inicial é gerada pela heurística NEH de Nawaz *et al.* (1983) considerada a melhor heurística para o CFSP. Os resultados do TS-M foram obtidos num computador Pentium 1.000 MHz. Os resultados do GASA foram obtidos num computador Athlon 1.400 MHz. Ambos são muito diferentes do computador usado pelo nosso AG, PC Dell com 3.200 MHz e 4 Gb de RAM.

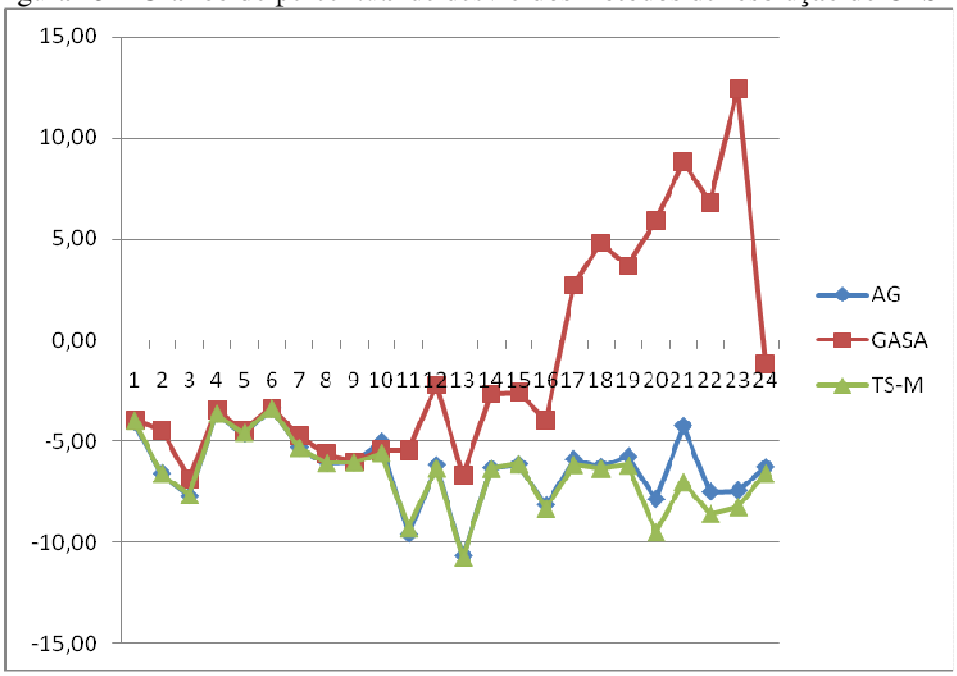
Tabela 11 – Resultados dos experimentos computacionais do AG para o CFSP.

Instância	n x m	RAJ	Pop. In.	Cruz.	Mut.	Mínimo	Desvio	GASA	TS-M	Tempo(s)
rec01	20x5	1590	2021	1526	1526	1526	**-4,03**	-3,96	-3,96	8,80
rec03	20x5	1457	1904	1361	1361	1361	**-6,59**	-4,46	**-6,59**	8,63
rec05	20x5	1637	1864	1511	1511	1511	**-7,70**	-6,90	-7,64	9,98
rec07	20x10	2119	2596	2042	2042	2042	**-3,63**	-3,45	**-3,63**	8,19
rec09	20x10	2141	2610	2042	2042	2042	**-4,62**	-4,48	-4,58	9,31
rec11	20x10	1946	2385	1881	1881	1881	**-3,34**	**-3,34**	**-3,34**	8,14
hel2	20x10	189	228	179	179	179	**-5,29**	-4,76	**-5,29**	7,94
rec13	20x15	2709	3231	2545	2545	2545	**-6,05**	-5,65	**-6,05**	9,55
rec15	20x15	2691	3384	2529	2529	2529	**-6,02**	**-6,02**	**-6,02**	7,76
rec17	20x15	2740	3206	2603	2603	2603	-5,00	-5,47	**-5,58**	8,88
rec19	30x10	3157	3918	2855	**2854**	2854	**-9,60**	-5,45	-9,25	15,47
rec21	30x10	3015	3807	2829	2829	2829	-6,17	-2,22	**-6,30**	11,81
rec23	30x10	3030	3897	2707	2707	2707	-10,66	-6,70	**-10,73**	15,83
rec25	30x15	3835	4907	3593	3593	3593	**-6,31**	-2,69	**-6,31**	12,45
rec27	30x15	3655	4779	3431	3431	3431	**-6,13**	-2,60	-6,10	13,98
rec29	30x15	3583	4986	3291	3291	3291	-8,15	-3,99	**-8,28**	14,92
rec31	50x10	4631	6383	4359	4359	4359	-5,87	2,72	**-6,13**	28,75
rec33	50x10	4770	6587	4472	4472	4472	-6,25	4,78	**-6,31**	29,11
rec35	50x10	4718	6930	4447	4447	4447	-5,74	3,67	**-6,17**	25,41
rec37	75x20	8979	12889	8273	8273	8273	-7,86	5,89	**-9,49**	48,91
rec39	75x20	9158	12841	8772	8772	8772	-4,21	8,80	**-6,99**	44,78
rec41	75x20	9344	13219	8643	8643	8643	-7,50	6,79	**-8,57**	59,20
hel1	100x10	780	1114	722	722	722	-7,44	12,44	**-8,21**	63,47
			Média				**-6,27**	**-1,18**	**-6,59**	**20,49**
			Mínimo				**-10,66**	**-6,90**	**-10,73**	**7,76**
			Máximo				**-3,34**	**12,44**	**-3,34**	**63,47**
			Total				**12**	**2**	**18**	

Fonte: Elaborado pelo Autor.

Estes dois métodos foram escolhidos para serem comparados com o AG devido aos seus testes terem sido realizados com as instâncias de Reeves (1995) e Heller (1960). A Tabela 11 mostra: as instâncias utilizadas nos experimentos; os valores de *m* e *n* para cada instância do problema; o valor da solução encontrada pelo algoritmo RAJ; o melhor valor encontrado por AG para cada um dos operadores *População Inicial* (Pop. In.), *Cruzamento* (Cruz.) e *Mutação* (Mut.); o valor mínimo de cada operador por instância; o percentual de desvio para os métodos de resolução GASA, TS-M e AG; o tempo de execução do AG; a média, o mínimo e o máximo dos desvios para cada operador; e o número de problemas que cada operador teve o melhor desempenho.

Figura 25 – Gráfico do percentual de desvio dos métodos de resolução do CFSP.



Fonte: Elaborado pelo Autor.

A análise da Tabela 11 produz as seguintes observações:

a) Nos 23 problemas testados o AG obteve melhor resultado em 5 problemas e em 7 problemas obteve resultado igual ao TS-M. Na média o AG foi 5,09 % superior ao GASA e 0,32 % inferior ao TS-M, considerado o melhor método de resolução do CFSP com o critério de desempenho sendo o *makespan*;

b) O tempo de execução do AG para os 23 problemas variou de 7,76 segundos a 63,47 segundos, com média de 20,49 segundos. Isto mostra a rapidez deste método em tratar problemas de médio e grande porte;

c) O melhor resultado do AG em comparação com o GASA foi de 19,88 % no problema *hel1*. Em relação ao TS-M foi de 0,35 % no problema *rec19*;

d) O AG foi melhor que o algoritmo de RAJ em todas as instâncias. Quanto a GASA, ele obteve melhor desempenho que o AG apenas na instância *rec17*, em todas as outras o AG foi superior ou na pior das hipóteses, AG e GASA tiveram o mesmo desempenho;

e) Os resultados do AG tendem a serem melhores que o GASA quando o número de tarefas e máquinas é grande;

f) Ainda nesta tabela é possível ver o bom desempenho do operador mutação na instância *rec19*;

g) O AG e o TS-M tiveram o menor valor máximo (-3,34 %). O TS-M teve o melhor mínimo -10,73 %, já o AG ficou muito próximo dele com -10,66 %;

h) O AG ficou muito próximo do desempenho de TS-M, conforme mostra o gráfico da Figura 25, dada anteriormente, para o percentual do desvio.

7 CONSIDERAÇÕES FINAIS

A metaheurística Algoritmo Genético abordada neste trabalho, com uso do procedimento células-tronco, mostrou que um AG que utiliza os princípios da diversificação e intensificação, de forma eficaz e eficiente, consegue obter bons resultados. Isto ficou comprovado com a implementação dos quatro tipos de *crossover*, atuando na diversificação, enquanto o operador mutação atua na intensificação. Este método foi aplicado com sucesso na resolução dos problemas de seqüenciamento com e sem restrição de espera. Neste trabalho, deu-se importância ao desenvolvimento de um método próprio com componentes originais sem usar recursos de inicialização eficiente e hibridização, para tornar o AG competitivo.

O AG mostrou ser mais eficaz e eficiente no CFSP sendo o *makespan* a função objetivo, com base nos resultados dos experimentos computacionais. Em se tratando de um método discreto aplicado na resolução do FSP, o AG também teve um excelente desempenho, pois chegou próximo ao melhor método de resolução do FSP que é o AGRu, considerado um método probabilístico.

O AG proposto mostrou que procedimentos bem sucedidos que acontecem no campo da Genética podem também ser adaptados e aplicados na teoria evolucionária dos algoritmos genético, como foi o caso de aplicação de células-tronco.

Propostas para futuros trabalhos:

- Incorporar novos operadores genéticos tais como cruzamento e mutação ao AG proposto. Uma sugestão é poder realizar mais de uma perturbação de cada vez e em intervalos de geração diferentes, dependendo da quantidade de tarefas e máquinas do problema;

- Como o AG obteve bons resultados para os dois tipos de problemas mesmo sem inicialização eficiente, uma proposta para melhorá-lo seria implementar uma inicialização eficiente que não comprometa a diversidade da população;

- Poderia ser testados outros valores para os parâmetros do AG quanto as combinações dos tipos de cruzamentos 1P, 2P, PM e 2B; e

- Aplicar o AG em outros problemas POCP e comparar o seu resultado com os melhores métodos desses problemas;

- Implementar o AG com auxílio do processamento distribuído, executando o método com populações inicias distintas para cada processador, podendo também ser executados em paralelo os operadores cruzamentos e mutação com o número de indivíduos da população podendo ser maior que 200.

REFERÊNCIAS

AARTS, E.; LENSTRA, J. K. **Local search in combinatorial optimization**. Wiley Interscience, Chichester, England. 1997.

ALDOWAISAN, T; ALLHVERDI, A. New heuristics for no-wait flowshops to minimize makespan. **Computers and Operations Research**, v.30, p.1219-1231, 2003.

ALDOWAISAN, T; ALLHVERDI, A. New heuristics for m-machine no-wait flowshop to minimize total completion time. **The International Journal of Management Science - Omega**, v.32, p.345-352, 2004.

BAKER, B. M. A genetic algorithm for the vehicle routing problem. **Computers & Operations Research**, v. 30, n. 5, p.787-800, 2003.

BLICKLE, T. Tournament Selection. Separata de: BACK, T.; FOGEL, D.; MICHALEWICZ, Z. (Ed.). **Handbook of Evolutionary Computation**. Oxford: Oxford University Press, 1997.

BLUM, C.; ROLI, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. **Computing Surveys**, v. 35, n. 3, p.268-308, 2003.

BRUCKER, P. **Scheduling Algorithms**. Springer-Verlag, Berlin. 1998.

CAMPBELL, H.G.; DUDEK, R.A.; SMITH, M.L. An heuristic algorithm for n job m machine sequencing problem. **Management Science**, v.16, p.630-637, 1970.

CHAKRAVARTHY, K.; RAJENDRAN, C. A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. **Production Planning and Control**, v.10, p.707-714, 1999.

CHEN, C. L.; VEMPATI, V.S.; ALJABER, N. An application of genetic algorithms for flow shop problems. **European Journal of Operational Research**, v.80, p.389-396, 1995.

CHEN, C. L.; NEPPALLI, R.V; ALJABER, N. Genetic algorithms applied to the continuous flow shop problem. **Computers and Industrial Engineering**, v.30, p.919-929, 1996.

CONWAY, R.W.; MAXWELL, W.L.; MILLER, L.W. **Theory of scheduling**. Reading, MA: Addison-Wesley; 1967.

DANNENBRING, D.G. An evaluation of flow shop sequencing heuristics. **Management Science**, v.23, p.1174-1182, 1977.

DRÉO, J.; PÉTROWSKI, A.; SIARRY, P.; TAILLARD, E. **Metaheuristics for Hard Optimization: Methods and Case Studies**. Springer, New York, 369 pp., 2006.

DUDEK, R.A.; TEUTON Jr., O.F. Development of m-stage decision rule for scheduling n jobs through m machines. **Operations Research**, v.12, p.471-497, 1964.

FINK, A.; VOβ, S. Solving the continuous flow-shop scheduling problem by metaheuristics. **European Journal of Operational Research**, v.151, p.400-414, 2003.

GANGADHARAN, R.; RAJENDRAN, C. Heuristic algorithms for scheduling in the no-wait flowshop. **International Journal of Production Economics**, v.32, p.285-290, 1993.

GAREY, M.R.; JOHNSON, D.S. **Computers and Intractability: a Guide to the Theory of NP-completeness**. W.H. Freeman and Company, San Francisco, CA, 338 pp., 1979.

GEN, M. **Genetic Algorithms and Their Applications**. Springer Handbook of Engineering Statistics. Springer London. p.749-773, 2006.

GOLDBERG, D.E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Addison Wesley, Reading, MA, 372 pp., 1989.

GRABOWSKI, J.; PEMPERA, J. Some local search algorithms for no-wait flow-shop problem with makespan criterion. **Computers and Operations Research**, v.32, p.2197-2212, 2005.

GRABOWSKI, J.; WODECKI, M. A very fast tabu search algorithm for the flow shop problem with makespan criterion. **Computers and Operations Research**, v.11, p.1891-1909, 2004.

GRAHAM, R.L.; LAWLER, E.L.; LENSTRA, J.K.; KAN, A.H.G.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 5:287–326, 1979.

GUPTA, J.N.D.; STAFFORD Jr, E.F. Flowshop scheduling research after five decades. **European Journal of Operational Research**, v.169, p.699–711, 2006.

HALL, N.G.; SRISKANDARAJAH, C. A survey of machine scheduling problems with blocking and no-wait in process. **Operations Research**, v.44, p.510-525, 1994.

HAUPT, R.L; HAUPT, S.E. **Pratical genetic algorithms**. John Wiley & Sons Inc., Hoboken, New Jersey, USA, 2ª ed., 253 pp., 2004.

HELLER, J. Some experiments for an M x J flow shop and its decision-theoretical aspects. **Operations Research**, v.8, p.178-184, 1960.

HOCHEDLINGER, K.; JAENISH, R. Nuclear Transplantation, Embryonic Stem Cells and the Potential for Cell Therapy". **N. England Journal of Medicine***,* 349:275-212, 2003.

HOLLAND, J. H. **Adaptation in natural artificial systems**. University of Michigan Press, Michigan, 211 pp., 1975.

JOHNSON, S.M. Optimal two- and three-stage production schedules with setup times included. **Naval Research logistics Quarterly** 1, p.61–68, 1954.

KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambrige, MIT Press, 1992.

LAWLER, E.L.; LENSTRAS, J.K.; RINNOOY KAN, A.H.G.; SHMOYS, D.B. **Sequencing and scheduling: Algorithms and complexity**. In: Graves, S.C. (Ed.), Handbooks in Operations Research and Management Science, Vol. 4. Elsevier Science Publishers, Amsterdam, pp. 445–552, 1993.

LENSTRA, J.K.; RINNOOY KAN, A.H.G.; BRUCKER, P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, p.343-362, 1977.

LOMNICKI, Z.A. A branch and bound algorithm for the exact solution of the three machine scheduling problem. **Operational Research Quarterly**, v.16, p.89-100, 1965.

MANNE, A. On the job-shop scheduling problem. **Operations Research**, v.8, p.219–223, 1960.

MITCHELL, M. **An introduction to genetic algorithms**. MIT Press, Cambridge, Massachusetts, USA, 158 pp., 1998.

MURATA, T.; ISHIBUCHI, H.; TANAKA, H. Genetic algorithms for flowshop scheduling problems. **Computers & Industrial Engineering**, v.30, p.1061-1071, 1996.

MUTH, J.; THOMPSON, G.L. **Industrial Scheduling**. Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

NAWAZ, M.; ENSCORE, E.; HAM, I. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. **OMEGA - The International Journal of Management Sciences**, v.11, p.91-95, 1983.

PANWALKAR, S.S.; WOOLLAM, C.R. Ordered flow shop problem with no in-process waiting: further results. **Journal of the Operational Research Society**, v.31, p.1039-1043, 1980.

PAPADIMITRIOU, C.H.; KANELLAKIS, P.C. Flowshop scheduling with limited temporary storage. **Journal of the Association for Computing Machinery**, v.27, p.533-549, 1980.

PERIN, E. C.; WILLERSON, J. T. Cell Therapy With Aldehyde Dehydrogenase Cells Shows Promise in Patients With Advanced Ischemic Heart Failure. **Heart Watch 2012 Spring**;2, 2012.

PERIN, E. C.; WILLERSON, J. T. Cell Therapy With Aldehyde Dehydrogenase Bright Cells May Benefit Patients With Critical Limb Ischemia. **Heart Watch 2011 Fall**;3, 2011.

RAJENDRAN, C. A no-wait flowshop scheduling heuristic to minimize makespan. **Journal of the Operational Research Society**, v.45, p.472-478, 1994.

RAJENDRAN, C.; CHAUDHURI, D. Heuristic algorithms for continuous flow-shop problem. **Naval Research Logistics**, v.37, p.695-705, 1990.

RAJENDRAN, C.; ZIEGLER, H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. **European Journal of Operational Research**, v.103, p.129-138, 1997.

REEVES, C. R. **Modern heuristic techniques for combinatorial problems**. Berkshire: McGrawHill Book Company, 1993.

REEVES, C. R. A genetic algorithm for flow shop sequencing. **Computers and Operations Research**, v.22, p.5-13, 1995.

RÖCK, H. The three-machine no-wait flowshop problem is NP-complete. **Journal of the Association for Computing Machinery**, v.31, p.336-345, 1984.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Two newrobust genetic algorithms for the flowshop scheduling problem. **The International Journal the Management Science (Omega)**, v.34, p.461-476, 2006.

SCHUSTER, C. J.; FRAMINAN, J. M. Approximative procedures for no-wait job shop scheduling. **Operations Research Letters**, v.31, p.308-318, 2003.

SILVA, J. L. C.; SOMA, N. Y. Um método heurístico aplicado no problema de programação flow shop permutacional. In: XXXVIII Simpósio Brasileiro de Pesquisa Operacional, 2006, Goiânia. Anais do XXXVIII SBPO, 2006.

SISSON, R. L. Methods of sequencing in job shops - a review. **Operations Research**, v.7, p.10–29, 1959.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, v.64, p.278-285, 1993.

T'KINDT, V.; BILLAUT, J. C. **Multi-criteria scheduling**. Springer-Verlag, Berlin, 1993.

VIANA, G. V. R. **Metaheurísticas e Programação Paralela em Otimização Combinatória**. UFC edições, 1998.

WAGNER, H.M. An integer linear-programming model for machine scheduling. **Naval Research Logistics Quarterly**, v.6, p.131–140, 1959.

WANG, L.; ZENG, D. Z. An effective hybrid optimization strategy for job-shop scheduling problemas. **Computers and Operations Research**, v.28, p.585-596, 2001.

WHITLEY, D. **A Genetic Algorithm Tutorial.** Fort Collins: Colorado University, 1993.