

27/08/99
BIBLIOTECA

**PROPOSIÇÃO DE UM MÉTODO METAHEURÍSTICO
HÍBRIDO ALGORITMO GENÉTICO-SIMULATED
ANNEALING PARA O PROBLEMA DE
PROGRAMAÇÃO DE OPERAÇÕES
FLOW SHOP PERMUTACIONAL**

Walther Rogério Buzzo



DEDALUS - Acervo - EESC



31100008542

Dissertação apresentada à Escola de Engenharia de São Carlos - Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Engenharia de Produção

Orientador: Prof. Dr. João Vitor Moccellin

São Carlos

1999

Class.	TESE
Gutt.	1004
Tombo	036/00

3110008542

S/S 1070560

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

B992p

Buzzo, Walther Rogério

Proposição de um método metaheurístico híbrido
algoritmo genético-*simulated annealing* para o problema
de programação de operações *flow shop* permutacional /
Walther Rogério Buzzo. -- São Carlos, 1999.

Dissertação (Mestrado) -- Escola de Engenharia de
São Carlos-Universidade de São Paulo, 1999.

Área: Engenharia de Produção.

Orientador: Prof. Dr. João Vitor Moccellin.

1. Programação da produção. 2. *Flow shop*
permutacional. 3. Metaheurísticas híbridas.
I. Título.

FOLHA DE APROVAÇÃO

Candidato: Engenheiro **WALTHER ROGÉRIO BUZZO**

Dissertação defendida e aprovada em 13-12-1999
pela Comissão Julgadora:



Prof. Titular **JOÃO VITOR MOCCELLIN (Orientador)**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)



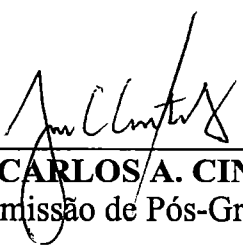
Prof. Doutor **EDSON WALMIR CAZARINI**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)



Prof. Doutor **FLAVIO CÉSAR FARIA FERNANDES**
(Universidade Federal de São Carlos - UFSCar)



Prof. Associado **RENATO VAIRO BELHOT**
Coordenador da Área de Engenharia de Produção



JOSÉ CARLOS A. CINTRA
Presidente da Comissão de Pós-Graduação da EESC

Dedico este trabalho aos meus pais,
Walther e Irene, pelo amor, paciência,
incentivo, oportunidade e sacrifícios.

Obrigado por tornarem este momento
possível.

Agradecimentos

* A Deus.

* Ao Professor e amigo Dr. João Vitor Moccellin, pela orientação fornecida durante todo o período de trabalho.

* Aos meus pais, Walther e Irene.

* Às minhas irmãs Adriana, Márcia e Patrícia.

* À Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro.

* A todos os professores e funcionários da Engenharia de Produção da EESC/USP, pela colaboração.

* Finalmente, gostaria de agradecer a todos os meus amigos da EESC/USP, os quais não poderei citar nominalmente, mas que sempre me incentivaram em todos os momentos. A todos o meu reconhecimento pelo apoio dado à transposição de mais esta etapa.

SUMÁRIO

LISTA DE FIGURAS	vi
LISTA DE TABELAS	viii
RESUMO	x
ABSTRACT	xi
1. PLANEJAMENTO E PROGRAMAÇÃO DA PRODUÇÃO	1
1.1. Introdução.....	1
1.2. Programação da Produção.....	3
1.3. Problemas de Programação de Operações em Máquinas.....	5
1.4. Objetivo e Estrutura do Trabalho.....	7
2. O PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES <i>FLOW SHOP</i> PERMUTACIONAL	9
2.1. Caracterização do Problema.....	9
2.2. Métodos Heurísticos Construtivos.....	11
2.3. Métodos Heurísticos Melhorativos.....	12
2.4. Métodos Heurísticos para a Solução do Problema de Programação de Operações <i>Flow Shop</i> Permutacional.....	13
3. MÉTODOS METAHEURÍSTICOS	25
3.1. Aspectos Gerais.....	25
3.2. Busca Tabu.....	26
3.3. <i>Simulated Annealing</i>	28
3.3.1. Aspectos Gerais.....	28
3.3.2. O Princípio de Funcionamento.....	29
3.3.3. Os Parâmetros Associados ao <i>Simulated Annealing</i>	30
3.3.4. Estrutura Básica do <i>Simulated Annealing</i>	33
3.4. Algoritmo Genético.....	34
3.4.1. Aspectos Gerais.....	34
3.4.2. O Princípio de Funcionamento.....	36

3.4.3. O Algoritmo Genético Simples.....	37
3.4.4. Estrutura Básica do Algoritmo Genético Simples	40
3.5. Métodos Metaheurísticos Híbridos.....	40
4. O MÉTODO METAHEURÍSTICO HÍBRIDO PROPOSTO.....	48
4.1. Aspectos Gerais	48
4.2. Parâmetros Quantitativos e Não-Quantitativos do Método Híbrido <i>HBGASA</i> ..	49
4.2.1. Soluções Iniciais	49
4.2.2. Vizinhança	49
4.2.3. Formas de Busca na Vizinhança.....	50
4.2.4. Condição de Parada.....	50
4.2.5. Operador de Reprodução.....	51
4.2.6. Operador de Cruzamento.....	51
4.2.7. Operador de Mutação.....	53
4.2.8. Temperatura Inicial T_1	54
4.2.9. Função de Resfriamento $F(T_k)$	54
4.2.10. Probabilidade de Aceitação $p(k)$	55
4.3. O Algoritmo Genético Puro	56
4.4. O Método <i>Simulated Annealing</i> Puro	57
4.5. O Método Híbrido <i>HBGASA</i>	58
5. EXPERIMENTAÇÃO COMPUTACIONAL E ANÁLISE DOS	
RESULTADOS OBTIDOS.....	61
5.1. Experimentação Computacional	61
5.2. Análise dos Resultados Obtidos	65
5.2.1. Método Híbrido <i>HBGASA</i> com operador de cruzamento de UM CORTE... 65	
5.2.2. Método Híbrido <i>HBGASA</i> com operador de cruzamento PMX.....	74
6. CONCLUSÕES.....	93
ANEXO 1	97

ANEXO 2 101

REFERÊNCIAS BIBLIOGRÁFICAS 135

LISTA DE FIGURAS

FIGURA 1: Esquema que relaciona as diversas classes de problema de programação de operações em máquinas (MACCARTHY & LIU (1993)).....	7
FIGURA 2: Exemplo de cruzamento de um corte	39
FIGURA 3: Mecanismo de funcionamento do operador mutação.....	39
FIGURA 4: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModSAfshopH X HBGASA 2070).....	71
FIGURA 5: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModSAfshopH X HBGASA 2075).....	72
FIGURA 6: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModSAfshopH X HBGASA 2085).....	73
FIGURA 7: Porcentagens de Sucesso para máquinas não-agrupadas (m = 4) pmxGA X ModSAfshopH X HBGASA 2030	81
FIGURA 8: Porcentagens de Sucesso para máquinas não-agrupadas (m = 7) pmxGA X ModSAfshopH X HBGASA 2030	81
FIGURA 9: Porcentagens de Sucesso para máquinas não-agrupadas (m = 10) pmxGA X ModSAfshopH X HBGASA 2030	82
FIGURA 10: Porcentagens de Sucesso para máquinas agrupadas pmxGA X ModSAfshopH X HBGASA 2030	82
FIGURA 11: Melhorias Relativas para máquinas não-agrupadas (m = 4) pmxGA X ModSAfshopH X HBGASA 2030	83
FIGURA 12: Melhorias Relativas para máquinas não-agrupadas (m = 7) pmxGA X ModSAfshopH X HBGASA 2030	84
FIGURA 13: Melhorias Relativas para máquinas não-agrupadas (m = 10) pmxGA X ModSAfshopH X HBGASA 2030	84
FIGURA 14: Melhorias Relativas para máquinas agrupadas pmxGA X ModSAfshopH X HBGASA 2030	85
FIGURA 15: Índice de Desempenho da Busca para máquinas não-agrupadas (m = 4) pmxGA X ModSAfshopH X HBGASA 2030	86

FIGURA 16: Índice de Desempenho da Busca para máquinas não-agrupadas ($m = 7$)	
pmxGA X ModSAfshopH X HBGASA 2030	86
FIGURA 17: Índice de Desempenho da Busca para máquinas não-agrupadas ($m=10$)	
pmxGA X ModSAfshopH X HBGASA 2030	87
FIGURA 18: Índice de Desempenho da Busca para máquinas agrupadas	
pmxGA X ModSAfshopH X HBGASA 2030	87
FIGURA 19: Tempos de Computação para máquinas não-agrupadas ($m = 4$)	
pmxGA X ModSAfshopH X HBGASA 2030	88
FIGURA 20: Tempos de Computação para máquinas não-agrupadas ($m = 7$)	
pmxGA X ModSAfshopH X HBGASA 2030	89
FIGURA 21: Tempos de Computação para máquinas não-agrupadas ($m = 10$)	
pmxGA X ModSAfshopH X HBGASA 2030	89
FIGURA 22: Tempos de Computação para máquinas agrupadas	
pmxGA X ModSAfshopH X HBGASA 2030	90

LISTA DE TABELAS

TABELA 1: Condição de Parada	51
TABELA 2: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 20$)	66
TABELA 3: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 30$)	67
TABELA 4: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 45$)	67
TABELA 5: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 60$)	68
TABELA 6: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 70$)	68
TABELA 7: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModSAfshopH X <i>HBGASA</i> 2070).....	71
TABELA 8: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModSAfshopH X <i>HBGASA</i> 2075).....	72
TABELA 9: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModSAfshopH X <i>HBGASA</i> 2085).....	73
TABELA 10: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 20$)	75
TABELA 11: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 30$)	75
TABELA 12: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 45$)	76
TABELA 13: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 60$)	76
TABELA 14: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e <i>HBGASA</i> ($T_1 = 70$)	77
TABELA 15: Comparação das Porcentagens de Sucesso da classe ($p = 30\%$).....	79
TABELA 16: Número de vitórias, empates e derrotas do método <i>HBGASA</i> ($p = 30\%$)	80

TABELA 17: Número de (vitórias + empates) do método <i>HBGASA</i> sobre o ModSAfshopH.....	92
TABELA 18: Porcentagens de Sucesso da classe (%) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas não-agrupadas).....	98
TABELA 19: Melhorias Relativas da classe (%) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas não-agrupadas)	98
TABELA 20: Índices de Desempenho da Busca da classe (em porcentagem $\times 10^{-3}$) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas não-agrupadas)	99
TABELA 21: Tempos Médios de Computação da classe (em segundos) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas não-agrupadas).....	99
TABELA 22: Porcentagens de Sucesso (%) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas agrupadas).....	100
TABELA 23: Melhorias Relativas (%) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas agrupadas).....	100
TABELA 24: Índices de Desempenho da Busca (em porcentagem $\times 10^{-3}$) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas agrupadas)....	100
TABELA 25: Tempos Médios de Computação (em segundos) para pmxGA, ModSAfshopH e <i>HBGASA</i> 2030 (máquinas agrupadas).....	100

RESUMO

BUZZO, W.R. (1999). *Proposição de um Método Metaheurístico Híbrido Algoritmo Genético - Simulated Annealing para o Problema de Programação de Operações Flow Shop Permutacional*. São Carlos, 1999. 140p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.

Este trabalho trata do problema de programação de operações *Flow Shop* Permutacional. Pelo fato de tal problema ser considerado *NP-hard*, diversos métodos heurísticos têm sido propostos com o objetivo de obter uma seqüência das tarefas que minimize a duração total da programação. Um dos tipos de métodos heurísticos consiste em melhorar soluções iniciais a partir de procedimentos de busca em vizinhança, tais como Algoritmo Genético (AG) e *Simulated Annealing* (SA). Nos últimos anos, métodos utilizando AG e SA têm sido apresentados para a solução de tal problema de programação da produção. Uma idéia interessante que tem despertado gradativa atenção refere-se ao desenvolvimento de métodos metaheurísticos híbridos utilizando Algoritmo Genético e *Simulated Annealing*. Assim, o objetivo é combinar as técnicas de tal forma que o procedimento resultante seja mais eficaz do que qualquer um dos seus componentes isoladamente. Neste trabalho é apresentado um método heurístico híbrido Algoritmo Genético - *Simulated Annealing* para minimizar a duração total da programação *flow shop* permutacional. Com o propósito de avaliar o desempenho do método híbrido, ele é comparado com métodos puros AG e SA que foram utilizados na sua concepção. Os resultados obtidos a partir de uma experimentação computacional são discutidos.

Palavras-chave: programação da produção, *flow shop* permutacional, metaheurísticas híbridas.

ABSTRACT

BUZZO, W.R. *Proposition of a Hybrid Metaheuristic Method Genetic Algorithm - Simulated Annealing for the Flow Shop Sequencing Problem*. São Carlos, 1999. 140p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.

This work deals with the Permutation Flow Shop Scheduling problem. Since this problem is NP-hard, many heuristic methods have been proposed for sequencing jobs in a flow shop with the objective of minimizing makespan. A class of such heuristics finds a good solution by improving initial sequences for the jobs through neighborhood search techniques as Genetic Algorithm (GA) and Simulated Annealing (SA). Recently, both GA and SA methods have been formulated for solving this scheduling problem. A promising approach for the problem is the formulation of hybrid metaheuristics by combining GA and SA techniques so that the consequent procedure is more effective than either pure GA or SA methods. In this work we present a hybrid Genetic Algorithm - Simulated Annealing heuristic for the minimal makespan flow shop sequencing problem. In order to evaluate the performance of the hybrid metaheuristic we compare it with pure GA and SA heuristics, which were used for the hybrid formulation. Results from computational experience are discussed.

Keywords: production scheduling, flow shop sequencing, hybrid metaheuristics.

CAPÍTULO 1

PLANEJAMENTO E PROGRAMAÇÃO DA PRODUÇÃO

1.1. Introdução

De acordo com RESENDE & SACOMANO (1991), sistemas de produção combinam fatores como materiais, trabalho e recursos de capital através de um processo organizado, com o objetivo de produzir mercadorias/serviços, de modo a atender as necessidades dos consumidores.

Conforme observado em FERNANDES (1991), a partir de um objetivo, deve-se estabelecer um plano para que o mesmo seja atingido. Porém, antes de colocá-lo em ação, é necessário organizar os recursos humanos e materiais, ou seja, estruturar, hierarquizar, atribuir responsabilidades, treinar etc. Simultaneamente, deve-se dirigir os recursos humanos para que eles ajam sobre os recursos materiais e controlar para que o rumo tomado seja compatível com o plano.

No contexto da administração da produção, este processo é realizado pelo Planejamento e Controle da Produção (PCP). O PCP está relacionado com uma série de atividades de longo, médio e curto prazos, fundamentais dentro do ciclo de produção de um produto.

Segundo ZACCARELLI (1986), o PCP é um conjunto de várias funções que tem por objetivo comandar e coordenar o processo produtivo. Essas funções são implementadas de diversas formas, dependendo do produto, do *layout* da fábrica etc.

SLACK *et al.* (1996) afirmam que o PCP requer a conciliação do fornecimento e da demanda em termos de: volume, tempo e qualidade. Para conciliar o volume e o tempo, são desempenhadas três atividades distintas, embora integradas:

- Carregamento: determinação do volume com o qual uma operação produtiva pode lidar. É a quantidade de trabalho alocada para um centro de trabalho.
- Seqüenciamento: determinação da prioridade de tarefas a serem desempenhadas. As prioridades dadas ao trabalho em uma operação são, freqüentemente, estabelecidas por um conjunto predefinido de regras, como por exemplo: prioridade ao consumidor, data prometida, *LIFO*, *FIFO* etc.
- Programação: decisão do tempo (momento) de início e fim para cada tarefa. Esta atividade é abordada com maiores detalhes no item seguinte.

De acordo com SILVA (1994), os tipos de problemas mais comuns encontrados no planejamento e controle das operações são:

→ Problemas de capacidade: devido a falta de mão-de-obra e equipamentos. Causam a queda no programa de produção, o que gera excessivo emprego de horas extras, atrasos nos programas de entrega, clientes insatisfeitos etc.

→ Problemas na programação/seqüenciamento da produção: causados pela falta de clareza em relação às prioridades das ordens de produção e ineficiência nas regras de seqüenciamento. Como consequência, a produção é interrompida por trabalhos que têm suas prioridades alteradas, atrasando os trabalhos que já estavam programados.

→ Reprogramação da produção: ocorre devido a fatores como problemas com operadores e máquinas, modificações no projeto, alterações nas quantidades solicitadas etc.

→ Controle de inventário ineficiente: neste caso, tem-se que ao mesmo tempo em que o inventário total está demasiadamente alto (matéria-prima, material em

processo e produtos acabados), existe a falta de estoque de itens individuais de matérias-primas necessárias para a produção. Conseqüentemente, tem-se o aumento dos custos com manutenção do inventário e atraso na produção devido à falta de matéria-prima.

→ Subutilização ou sobrecarga de trabalhadores e equipamentos: devido às excessivas variações nas quantidades e tipos de produtos pedidos, interrupções nos trabalhos, quebra de equipamentos etc.

→ Problemas de qualidade: são encontrados nos componentes produzidos e nos produtos montados. Isto causa reprogramações, gerando atrasos na programação e nas entregas.

→ Erros de projeto/especificações: muitas vezes são detectados quando o produto já está sendo fabricado. Isto gera a necessidade de correções, que levam ao retrabalho ou sucateamento .

→ Problemas de manutenção: a quebra de máquinas causa reprogramação, atraso nas entregas, sobrecarga dos centros de trabalho etc. É necessário, portanto, o desenvolvimento de um programa de manutenção eficiente, que envolva reparos preventivos quando o equipamento não estiver sendo solicitado.

1.2. Programação da Produção

BURBIDGE (1986) define programação como:

i) a determinação de quando e onde cada operação necessária para a fabricação de um produto deve ser realizada ou

ii) a determinação das datas em que se deve iniciar e/ou completar cada evento ou operação que compõe um procedimento.

Conforme observado em SILVA (1994), a programação no nível da liberação da produção é uma atividade de curto prazo e determina para cada ordem de fabricação, quando é necessário iniciar a fabricação e quanto é preciso trabalhar em cada uma das operações planejadas.

Isto se torna possível pelo conhecimento do *lead time* de cada componente, o qual compreende os tempos de processamento e de montagem de cada operação e os tempos de movimentação e espera entre as operações.

③ ⇒ A programação da produção envolve fatores externos e internos, cada um deles conduzindo a estratégias diferentes.

Em relação aos aspectos externos, é importante notar que a programação procura acomodar a influência da demanda, representada pelas solicitações dos clientes, em quantidade e prazo. Estabelecer prazos consiste em subtrair de uma data de término (de um prazo de entrega), os tempos de execução (duração) e as tolerâncias de fabricação. Assim, são obtidas as datas de início da fabricação dos produtos ou de execução das atividades.

Os aspectos internos da programação estão ligados à utilização eficiente dos recursos (produtividade). Têm como estratégia, a utilização eficiente da capacidade, através da coordenação das atividades que ocorrem internamente. Tais atividades referem-se à ordenação das tarefas, ou seja, quais devem ser realizadas (viáveis e disponíveis) e em que ordem. Este tipo de programação busca obter a melhor eficiência dos recursos, como por exemplo, a máxima utilização das máquinas.

Em outras palavras, a programação da produção orientada externamente busca satisfazer objetivos ligados ao nível de serviços ao cliente, ao passo que a programação orientada internamente procura atingir a máxima produtividade dos recursos. A estratégia utilizada na programação da produção pode variar de empresa para empresa, mas alguns objetivos básicos são preservados. Os principais são:

- ⇒ entregar os produtos fabricados nas datas compromissadas ou estabelecidas;
- ⇒ distribuir a carga de trabalho de forma a obter a máxima utilização dos recursos;
- ⇒ garantir que toda matéria-prima e componentes comprados estejam disponíveis quando forem solicitados pela fabricação;
- ⇒ prever e evitar grande concentração de trabalho em poucas máquinas (gargalos de produção);
- ⇒ prever ociosidade da capacidade produtiva e

⇒ estabelecer seqüências de produção que minimizem o tempo de equipamento sem trabalho.

O método de programação mais comumente utilizado é o Gráfico de Gantt. Porém, diversas técnicas e métodos de programação da produção foram desenvolvidos com fins específicos, dentre eles o Diagrama de Montagem, as técnicas de Redes CPM e PERT e outros métodos que determinam o melhor seqüenciamento da produção.

⇒ 1.3. Problemas de Programação de Operações em Máquinas

De acordo com SLACK *et al.* (1996), a atividade de programação é uma das mais complexas no gerenciamento da produção. Os programadores têm que lidar com diversos tipos diferentes de recursos simultaneamente. As máquinas possuem diferentes capacidades. O pessoal tem diferentes habilidades. E o mais importante: o número de programações possíveis cresce rapidamente à medida em que o número de atividades e processos aumenta.

Existem, portanto, diversos fatores que tornam a programação da produção um problema difícil. A característica fundamental do problema é a sua natureza combinatorial. Isto significa que o número de possíveis soluções cresce exponencialmente em várias dimensões, de acordo com a quantidade de tarefas, operações, máquinas, pessoal etc.

Por exemplo, a programação de 12 tarefas com 6 operações apresenta $(12!)^6$ soluções possíveis em um *Job Shop* tradicional, sem máquinas alternativas. E tal situação piora quando existem máquinas alternativas que realizam a mesma operação ou máquinas que podem realizar várias operações.

Se relacionarmos isso a uma situação real, onde pode existir, por exemplo, 100 trabalhos e 30 máquinas, em roteiros em que cada trabalho use 5 máquinas diferentes, podemos notar que a tarefa de programação torna-se muito complicada. Dentre um grande número de programações, há várias opções aceitáveis, há vários roteiros e seqüências apropriadas para qualquer conjunto de trabalho.

A tarefa de programação tem que ser freqüentemente repetida para permitir resposta às variações de mercado e às mudanças no *mix* de produtos.

Dentro de uma situação de programar as operações nas máquinas disponíveis, surgem problemas complexos. Dentro destes problemas, as restrições tecnológicas e a medida de desempenho da programação devem ser especificadas.

As restrições tecnológicas são determinadas principalmente pelo fluxo das tarefas nas máquinas. Dentro deste contexto, MACCARTHY & LIU (1993) classificaram os problemas de programação de operações em:

➔ **Job Shop**: cada tarefa possui um roteiro específico de processamento nas máquinas;

➔ **Flow Shop**: todas as tarefas possuem o mesmo fluxo de processamento nas máquinas;

➔ **Open Shop**: não há roteiros de processamento específicos ou preestabelecidos para as tarefas a serem processadas nas máquinas;

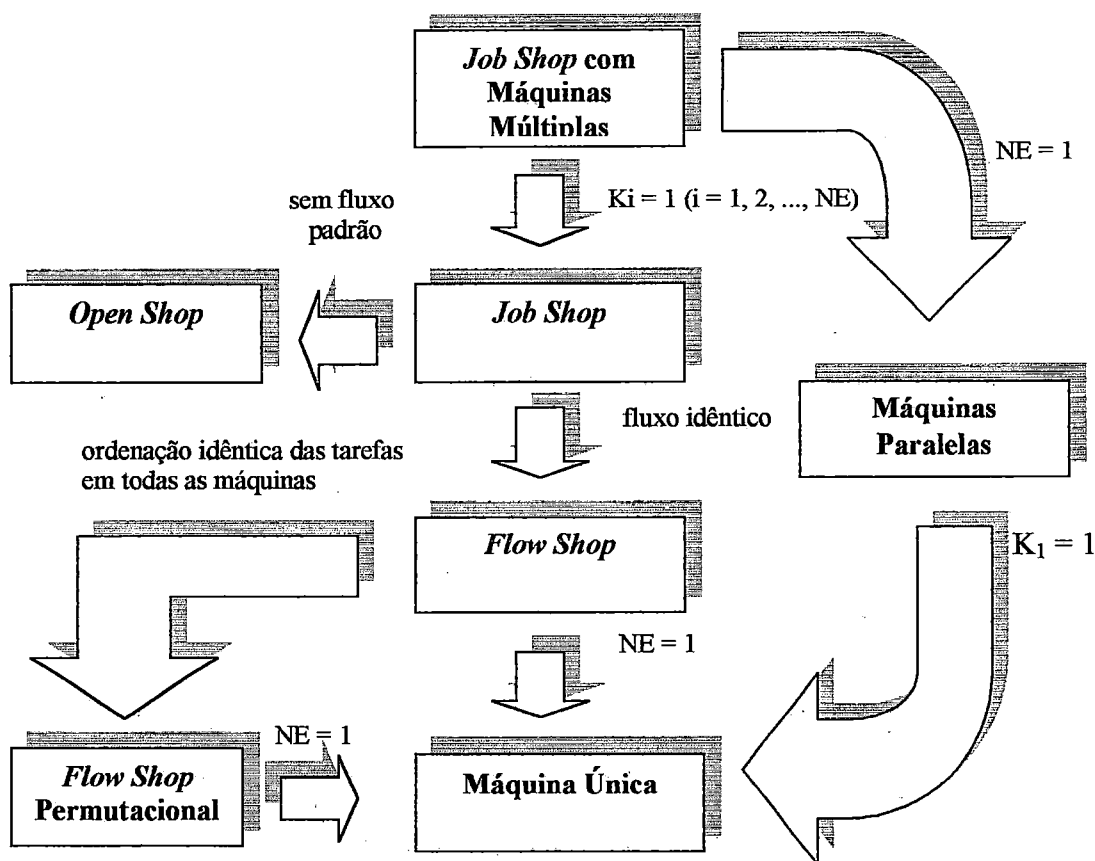
➔ **Flow Shop Permutacional**: é um *Flow Shop* no qual em cada máquina a seqüência de processamento das tarefas é a mesma;

➔ **Máquina Única**: existe uma única máquina disponível;

➔ **Máquinas Paralelas**: são disponíveis duas ou mais máquinas, geralmente idênticas, que podem executar qualquer tarefa e

➔ **Job Shop com Máquinas Múltiplas**: é um *Job Shop* no qual existem k_i máquinas idênticas em cada estágio i ($i = 1, 2, \dots, NE$), sendo que, em cada estágio, cada tarefa é processada por somente uma máquina.

A Figura 1 ilustra a relação entre as diversas classes de problemas de programação de operações.



K_i = número de máquinas do estágio i

NE = número de estágios

FIGURA 1: Esquema que relaciona as diversas classes de problemas de programação de operações em máquinas (MACCARTHY & LIU (1993)).

1.4. Objetivo e Estrutura do Trabalho

O presente trabalho tem como objetivo propor um método heurístico híbrido que combina dois métodos metaheurísticos - denominados Algoritmo Genético e *Simulated Annealing* - para a solução do Problema de Programação de Operações *Flow Shop* Permutacional. O método híbrido proposto é comparado com os

metaheurísticos Algoritmo Genético e *Simulated Annealing* puros e o seu desempenho é avaliado por meio de uma experimentação computacional.

O trabalho foi dividido da seguinte forma:

☞ **Capítulo 1:** aborda o Planejamento e Programação da Produção e apresenta o objetivo do trabalho.

☞ **Capítulo 2:** define-se o problema de Programação de Operações *Flow Shop* Permutacional e são apresentados os métodos heurísticos mais conhecidos reportados na literatura, para a solução deste problema.

☞ **Capítulo 3:** são apresentados os métodos metaheurísticos Busca Tabu, Algoritmo Genético e *Simulated Annealing*, destacando-se os dois últimos.

☞ **Capítulo 4:** é dedicado aos métodos metaheurísticos híbridos reportados na literatura, que combinam as técnicas de Busca Tabu, Algoritmo Genético e *Simulated Annealing*.

☞ **Capítulo 5:** é apresentado o método metaheurístico híbrido proposto neste trabalho.

☞ **Capítulo 6:** é descrita a experimentação computacional e apresentada a análise dos resultados obtidos.

☞ **Capítulo 7:** relatam-se as conclusões finais e propostas para trabalhos futuros.

No **Anexo 1** são encontrados os resultados obtidos na experimentação computacional para o método desenvolvido de melhor desempenho.

Os programas computacionais do método metaheurístico híbrido, Algoritmo Genético puro e *Simulated Annealing* puro são apresentados no **Anexo 2**.

CAPÍTULO 2

O PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES *FLOW SHOP* PERMUTACIONAL

2.1. Caracterização do Problema

O problema de Programação de Operações em um ambiente *Flow Shop* é um problema de programação da produção no qual cada uma de um conjunto de n tarefas deve ser processada por um conjunto de m máquinas distintas, tendo o mesmo fluxo de processamento nas máquinas. Quando em cada máquina a ordem de processamento das tarefas é a mesma, tem-se o *Flow Shop* Permutacional. Tal problema é típico de instalações de manufatura onde as tarefas (peças) são movidas de uma máquina para outra através de equipamentos de movimentação de materiais.

A solução do problema consiste em se determinar dentre as $(n!)$ seqüências possíveis das tarefas, aquela que otimiza uma determinada medida de desempenho da programação, geralmente associada ao fator tempo. Dentre as diferentes medidas de desempenho, as mais utilizadas são aquelas que buscam minimizar:

- a duração total da programação (*makespan*);

- o tempo médio de fluxo ou tempo médio de permanência das tarefas (*mean flow-time*);
- o atraso médio na execução das tarefas (*mean tardiness*) e
- o atraso máximo na execução das tarefas (*maximum tardiness*).

Usualmente, procura-se minimizar a duração total da programação; isto é, o intervalo de tempo entre o início de execução da primeira tarefa na primeira máquina e o término de execução da última tarefa na última máquina.

As hipóteses usuais do problema de Programação de Operações *Flow Shop* são:

- 1) cada máquina está disponível continuamente, sem interrupções;
- 2) cada máquina pode processar apenas uma tarefa de cada vez;
- 3) cada tarefa pode ser processada por uma máquina de cada vez;
- 4) os tempos de processamento das tarefas nas diversas máquinas são determinados e fixos;
- 5) as tarefas têm a mesma data de liberação, a partir da qual, qualquer uma pode ser programada e executada;
- 6) os tempos de preparação das operações nas diversas máquinas são incluídos nos tempos de processamento e independem da seqüência de operações em cada máquina e
- 7) as operações nas diversas máquinas, uma vez iniciadas não devem ser interrompidas.

Na teoria que estuda a complexidade dos problemas de natureza combinatorial, o problema em questão é classificado como *NP-hard* (GAREY *et al.*, 1976), de forma que pode ser resolvido eficientemente de maneira ótima somente em casos de pequeno porte. Resolver um problema de otimização combinatorial consiste em se encontrar a melhor solução ou a solução ótima dentre um número finito de soluções possíveis. Por exemplo, um problema que envolve 10 tarefas apresenta 3.628.800 programações possíveis.

Um dos pioneiros que estudou o problema foi JOHNSON (1954), que apresentou um algoritmo eficiente computacionalmente que fornece a solução ótima para o caso de duas ou três máquinas (este último caso, sob certas restrições). Por este algoritmo, o início da seqüência ótima é composto pelas tarefas que apresentam tempos de processamento menores na primeira máquina; o final da seqüência é composto pelas tarefas que têm os tempos de processamento menores na segunda máquina.

Nas últimas três décadas, um extenso esforço de pesquisa tem sido dedicado ao problema. Técnicas de Programação Matemática, tais como Programação Linear Inteira (SELEN & HOTT (1986)) e técnicas de enumeração do tipo *branch-and-bound* (IGNALL & SCHRAGE (1965)), têm sido empregadas para a solução ótima do problema. Entretanto, tais técnicas não são eficientes em termos computacionais, em problemas de médio e grande porte. Assim, muitos métodos heurísticos têm sido propostos para a solução do problema de Programação de Operações *Flow Shop* Permutacional.

A disponibilidade de recursos computacionais mais poderosos tem propiciado muita pesquisa e, conseqüentemente, o desenvolvimento de métodos heurísticos muito eficientes. Os métodos heurísticos devem ser procedimentos mais flexíveis e simples que os métodos exatos de solução, permitindo-lhes abordar modelos mais complexos. Os métodos exatos geralmente exigem simplificações nos modelos, tornando-os menos representativos do problema real.

Um método heurístico não pode garantir a qualidade da solução, sendo aceitável somente se os resultados forem satisfatórios quanto ao esforço computacional e quanto à proximidade da solução ótima. Em geral, um método heurístico é fortemente dependente das características do problema e da distribuição dos dados do mesmo. Estes métodos podem ser classificados em dois grupos: **métodos construtivos e métodos melhorativos.**

2.2. Métodos Heurísticos Construtivos

Geram uma única seqüência, a qual é adotada como solução final do problema. Tal seqüência pode ser obtida:

- diretamente a partir da ordenação das tarefas segundo índices de prioridade calculados em função dos tempos de processamento das tarefas, como por exemplo: PALMER (1965);
- escolhendo-se a melhor seqüência das tarefas a partir de um conjunto de seqüências também obtidas utilizando-se índices de prioridade associados às tarefas: CAMPBELL, DUDEK & SMITH (1970) e HUNDAL & RAJGOPAL (1988) ou
- a partir da geração sucessiva de seqüências parciais das tarefas (sub-seqüências) até a obtenção de uma seqüência completa através de algum critério de inserção de tarefas, como por exemplo: NEH (NAWAZ, ENSCORE & HAM (1983)).

2.3. Métodos Heurísticos Melhorativos

Obtém-se uma solução inicial e posteriormente através de algum procedimento iterativo (geralmente envolvendo trocas de posições das tarefas na seqüência) busca-se obter uma seqüência das tarefas melhor que a atual quanto à medida de desempenho adotada.

Na categoria dos métodos melhorativos destacam-se os procedimentos de busca em vizinhança, como por exemplo DANNENBRING (1977), considerado um método de busca simples. Recentemente, foram desenvolvidos métodos de busca em vizinhança de maior complexidade (Busca Tabu e *Simulated Annealing*) que têm sido alvo de grande interesse na comunidade científica em função de aplicações bem sucedidas reportadas na literatura. Exemplos de aplicações dessas técnicas para o problema *Flow Shop* Permutacional são encontrados em: OSMAN & POTTS (1989); WIDMER & HERTZ (1989); OGBU & SMITH (1990); TAILLARD (1990); MOCCELLIN (1995); ISHIBUCHI, MISAKI & TANAKA (1995); ZEGORDI, ITOH & ENKAWA (1995); PARK & KIM (1998) e MOCCELLIN & NAGANO (1998).

Outra técnica que pode ser considerada do tipo melhorativo, denominada Algoritmo Genético, tem despertado interesse pela sua capacidade de solução de problemas de natureza combinatorial. REEVES (1995) utilizou o Algoritmo Genético para a minimização do *makespan* em um *Flow Shop* Permutacional.

2.4. Métodos Heurísticos para a Solução do Problema de Programação de Operações *Flow Shop* Permutacional

Nesta seção são descritos sucintamente os métodos heurísticos mais conhecidos reportados da literatura. Antes, a seguinte notação deve ser observada:

- Seja $J = \{J_1, J_2, \dots, J_i, \dots, J_n\}$ o conjunto das n tarefas que devem ser processadas, na mesma seqüência, por um conjunto de m máquinas distintas. O tempo de processamento da tarefa J_i na máquina k é p_{ki} ($i = 1, 2, \dots, n$ e $k = 1, 2, \dots, m$). A tarefa que não tiver operação em uma determinada máquina tem o tempo de processamento igual a zero.

PALMER (1965) desenvolveu um índice denominado *Slope Index*, o qual estabelece a seqüência de processamento das tarefas nas máquinas. Tal índice é calculado de modo que as tarefas cujos tempos de processamento tendem a crescer na seqüência das máquinas devem receber prioridade na programação, ocupando as primeiras posições na ordem de execução das tarefas. Para uma tarefa J_i , o *Slope Index* é dado pela expressão:

$$S_i = \sum_{k=1}^m (2k - m - 1) \cdot p_{ki}$$

para: $i = 1, 2, \dots, n$.

A seqüência de programação é estabelecida de acordo com a ordenação não-crescente dos índices.

CAMPBELL, DUDEK & SMITH (1970) propuseram um procedimento chamado CDS, que é basicamente uma generalização do algoritmo de JOHNSON para a solução ótima do problema com duas máquinas. O CDS gera $(m-1)$ subproblemas artificiais de duas máquinas a partir do problema original de m

máquinas e, em seguida, usa a regra de JOHNSON em $(m-1)$ estágios, onde cada estágio corresponde a um problema com duas máquinas e com tempos de processamento “artificiais” p_{1i}' e p_{2i}' ($i = 1, 2, \dots, n$). No estágio 1, $p_{1i}' = p_{1i}$ e $p_{2i}' = p_{mi}$, isto é, a regra de JOHNSON é aplicada considerando-se a primeira e a última máquinas, desprezando-se as demais. No estágio 2, temos $p_{1i}' = p_{1i} + p_{2i}$ e $p_{2i}' = p_{(m-1)i} + p_{mi}$, ou seja, aplica-se a regra de JOHNSON à soma dos tempos de processamento da primeira com a segunda máquina e da penúltima com a última máquina. No estágio t , os tempos de processamento artificiais serão determinados pelas expressões:

$$p_{1i}' = \sum_{k=1}^t p_{ki} \quad \text{e} \quad p_{2i}' = \sum_{k=1}^t p_{(m-k+1)i}$$

Em cada estágio, a seqüência de tarefas obtida pela regra de JOHNSON é usada para calcular a duração total da programação do problema original. A seqüência que fornece a menor duração, torna-se a solução do problema.

GUPTA (1971) reconheceu que o algoritmo de JOHNSON para o problema com duas ou três máquinas é, na verdade, um método de ordenação a partir da definição de um fator para cada tarefa, seqüenciando-as de acordo com a ordem crescente destes fatores. A função de indexação foi generalizada para o caso de $m \geq 4$ máquinas, definindo para cada tarefa, o seguinte índice:

$$f(i) = A / \min_{1 \leq k \leq m-1} (p_{ki} + p_{k+1,i})$$

- onde:
- $i = 1, 2, \dots, n$ e
 - $A = 1$, se $p_{ki} \leq p_{1i}$ ou $A = -1$ se $p_{ki} > p_{1i}$.

DANNENBRING (1977) propôs um procedimento chamado *Rapid Access* (RA), que combina as vantagens do *Slope Index* de PALMER e do método CDS.

Diferentemente do CDS, o método RA resolve um único problema artificial de duas máquinas, onde os tempos de processamento são determinados pelas seguintes expressões:

$$p_{1i}' = \sum_{k=1}^m (m - k + 1) \cdot p_{ki} \quad \text{e} \quad p_{2i}' = \sum_{k=1}^m (k) \cdot p_{ki}$$

para: $i = 1, 2, \dots, n$.

Utilizando a solução fornecida pelo método RA, DANNENBRING propôs dois procedimentos melhorativos: o *Rapid Access with Close Order Search* (RACS) e o *Rapid Access with Extensive Search* (RAES), que buscam entre as seqüências vizinhas da solução inicial, uma seqüência que forneça uma duração total menor. Seqüência vizinha é uma nova seqüência formada pela transposição de um par de tarefas adjacentes, a partir de uma seqüência atual.

No método RACS são examinadas as $(n-1)$ seqüências vizinhas, adotando-se como solução a seqüência de menor *makespan*, desde que este seja menor que o *makespan* da solução inicial. O método RAES, ao invés de finalizar o procedimento de melhoria após examinar as $(n-1)$ seqüências vizinhas da solução inicial, usa a melhor seqüência vizinha para gerar suas próprias vizinhas, até encontrar uma seqüência cujas vizinhas não apresentem uma menor duração total da programação.

NAWAZ, ENSCORE & HAM (1983) desenvolveram o algoritmo NEH, baseado na hipótese de que a tarefa com maior tempo de processamento nas m máquinas deve ter maior prioridade de programação em relação às tarefas com menor tempo total de processamento. O funcionamento básico do algoritmo NEH é o seguinte:

Inicialmente, calcula-se a soma dos tempos de processamento para cada tarefa nas m máquinas. As n tarefas são, então, ordenadas em função dos valores não-crescentes das somas dos tempos de processamento.

As duas tarefas com maior tempo total de processamento são selecionadas a partir das n tarefas. A melhor seqüência parcial para essas duas tarefas é encontrada pela busca exaustiva, isto é, considerando-se os dois possíveis seqüenciamentos

parciais. As posições destas duas tarefas com respeito às demais são fixadas no restante do algoritmo.

Em seguida, a tarefa com o terceiro tempo total de processamento é selecionada e as três seqüências parciais possíveis entre essas tarefas são testadas, em que tal tarefa é colocada no começo, meio e final da seqüência parcial encontrada no primeiro passo. A melhor seqüência parcial fixará a posição relativa destas três tarefas para os passos restantes. O processo se repete até que todas as tarefas sejam seqüenciadas.

HUNDAL & RAJGOPAL (1988) constataram que o algoritmo de PALMER ignora a máquina $(m+1)/2$ quando m é ímpar, o que pode afetar a qualidade da solução, particularmente quando o número de tarefas n é grande. Eles desenvolveram, então, uma extensão do algoritmo de PALMER, que é considerada a partir de dois novos conjuntos de fatores dados pelas expressões:

$$S_i = \sum_{k=1}^m (2k - m) \cdot p_{ki} \quad \text{e} \quad S_i = \sum_{k=1}^m (2k - m - 2) \cdot p_{ki}$$

para: $i = 1, 2, \dots, n$.

Observa-se que duas outras seqüências são obtidas neste método, sendo selecionada a melhor delas.

WIDMER & HERTZ (1989) desenvolveram o método melhorativo SPIRIT (*Sequencing Problem Involving a Resolution by Integrated Tabu Search Techniques*) que, a partir de uma solução inicial, procura obter uma solução melhor, examinando uma série de seqüências vizinhas. O SPIRIT apresenta duas etapas básicas: na primeira etapa, a solução inicial é obtida através de uma analogia com o problema do caixeiro-viajante, onde a distância entre duas tarefas (cidades) J_u e J_v é calculada pela seguinte expressão:

$$d_{uv} = p_{1u} + \sum_{j=2}^m (m - j) \cdot |p_{ju} - p_{j-1,v}| + p_{mv}$$

- onde:
- m = número de máquinas e
 - p_{ji} = tempo de processamento da tarefa J_i na máquina j .

Na segunda etapa, a solução inicial é melhorada utilizando-se a técnica denominada Busca Tabu. Esta técnica é apresentada com mais detalhes no próximo capítulo.

OSMAN & POTTS (1989) propuseram alguns métodos heurísticos para o problema de Programação de Operações *Flow Shop* Permutacional baseados na técnica denominada *Simulated Annealing* (SA). A idéia do SA tem origem na termodinâmica e na metalurgia: quando o ferro fundido é lentamente resfriado, tende a se solidificar em uma estrutura de mínima energia. KIRKPATRICK *et al.* (1983) foram os primeiros a utilizar o *Simulated Annealing* para resolver problemas de otimização combinatorial. Tal técnica também é apresentada com maiores detalhes no próximo capítulo.

HO & CHANG (1991) apresentaram um algoritmo melhorativo em que a solução inicial é obtida empregando-se métodos já existentes, como *Slope Index*, *CDS* e *Rapid Access*. Na segunda fase, os autores propuseram um procedimento de melhoria da solução inicial baseado nas relações entre os tempos de processamento das tarefas, que são consideradas em pares, ou mais especificamente, utilizando-se as seguintes diferenças:

$$d_{ij}^k = p_{k+1,i} - p_{kj}$$

- para :
- $i, j = 1, 2, \dots, n$;
 - $k = 1, 2, \dots, (m-1)$ e
 - $i \neq j$;
- onde:
- $p_{k+1,i}$ é o tempo de processamento da tarefa J_i na máquina $(k+1)$ e
 - p_{kj} é o tempo de processamento da tarefa J_j na máquina k .

MOCCELLIN (1992) desenvolveu um novo método heurístico - denominado FSHOPH - semelhante ao SPIRIT. O método FSHOPH também usa a analogia com o problema do caixeiro-viajante, mas com uma diferença relevante: na obtenção da solução inicial do problema, a expressão referente à “distância” entre duas tarefas é diferente daquela empregada por WIDMER & HERTZ. A duração total da programação $M(S)$ para uma seqüência qualquer S pode ser dada pela seguinte expressão:

$$M(S) = \sum_{j=1}^n p_{mj} + \sum_{j=0}^{n-1} X_{j+1}^m, \quad \text{onde:}$$

- p_{mj} : é o tempo de processamento, na última máquina, da tarefa que ocupa a j -ésima posição na seqüência S ;
- X_{j+1}^m : é o intervalo de tempo entre o término da operação da tarefa que ocupa a j -ésima posição na seqüência S e o início da operação da tarefa que ocupa a posição $(j+1)$ de S , na última máquina e
- $j = 0$: corresponde a uma tarefa fictícia com tempos de processamento nulos que ocupa sempre a primeira posição em qualquer seqüência das n tarefas.

Se considerarmos X_{j+1}^m como sendo a distância entre duas tarefas que ocupam as posições j e $(j+1)$ da seqüência S , estabelece-se uma analogia entre os problemas do caixeiro-viajante e de Programação de Operações *Flow Shop* Permutacional. A “rota mínima que interliga” as n tarefas corresponde à seqüência que minimiza a duração total da programação.

Os valores de X_{j+1}^m não são conhecidos previamente (dependem da seqüência S considerada). As distâncias entre as tarefas são estimadas a partir dos tempos de processamento das tarefas, de modo que a solução do problema do caixeiro-viajante corresponda a uma solução heurística do problema original de Programação de Operações.

O autor observou que, para o caso geral de n e m quaisquer, e sendo

X_{j+1}^{k+1} , o tempo de espera na máquina (k+1), entre as operações que ocupam as posições j e (j+1) da seqüência S, um limitante superior desse tempo de espera é dado pela seguinte expressão:

$$UBX_{j+1}^{k+1} = \max \{0, UBX_{j+1}^k + (p_{k,j+1} - p_{k+1,j})\}, \text{ onde:}$$

- $UBX_{j+1}^1 = 0$ (não existe tempo de espera entre tarefas sucessivas na primeira máquina);
- UBX_{j+1}^{k+1} : limitante superior de X_{j+1}^{k+1} ;
- UBX_{j+1}^k : limitante superior de X_{j+1}^k ;
- $p_{k,j+1}$: tempo de processamento na máquina k da tarefa que ocupa a posição (j+1) e
- $p_{k+1,j}$: tempo de processamento na máquina (k+1) da tarefa que ocupa a posição j.

Uma vez definido um limitante superior para X_{j+1}^m , pode-se estabelecer um limitante superior para a duração total da programação, dado pela expressão:

$$UBM(S) = \sum_{j=1}^n p_{mj} + \sum_{j=0}^{n-1} UBX_{j+1}^m$$

Considerado um par qualquer de tarefas J_u e J_v (J_u precedendo diretamente J_v) de um conjunto de n tarefas, um limitante superior para o tempo de espera, na última máquina, entre as tarefas J_u e J_v , independentemente da posição ocupada por J_u , é obtido através da expressão:

$$UBG_{uv} = \max \{0, UBX_{uv}^{m-1} + (p_{m-1,v} - p_{m,u})\}$$

No primeiro passo do método FSHOPH, a “distância” entre as tarefas J_u e J_v é dada por UBG_{uv} , e procura-se a “rota” (ou seja, a seqüência das n tarefas) que

minimiza o limitante superior da duração total da programação UBM(S). No segundo passo, como no caso do método SPIRIT, a solução inicial é melhorada através da Busca Tabu.

NAGANO (1995) apresentou novos procedimentos de Busca Tabu para a resolução do problema de Programação de Operações *Flow Shop* Permutacional. Foram propostos nove procedimentos alternativos de Busca Tabu, a partir da combinação de diferentes estruturas de vizinhança (*Interchange*, *Shift* e *Adjacent Neighbourhood*), de formas de busca na vizinhança distintas (total e parcial aleatória) e duas condições de parada (Nbmax e Msatisf). Partindo de uma mesma solução inicial (fornecida pelo método FSHOPH), os desempenhos dos nove procedimentos propostos foram avaliados, em termos de esforço computacional e qualidade da solução. Uma contribuição relevante do referido trabalho diz respeito à determinação de duas novas funções que fornecem o número de seqüências vizinhas a serem avaliadas para as vizinhanças *Interchange* e *Shift*. Estas funções estabelecem a parcela do total de vizinhos que devem ser analisados (e não a vizinhança total). Conforme o valor de n aumenta, o número de seqüências a serem avaliadas converge para um limite computacionalmente adequado.

As funções definidas para as respectivas vizinhanças são dadas pelas expressões:

$$\Rightarrow \textit{Interchange Neighbourhood} : f(n) = n(n - 1) / 2 , \text{ para } n \leq 40 \quad \text{ou}$$

$$f(n) = 1780 - 1000 e^{-0,03(n - 40)} , \text{ para } n > 40$$

$$\Rightarrow \textit{Shift Neighbourhood} : f(n) = (n - 1)^2 , \text{ para } n \leq 30 \quad \text{ou}$$

$$f(n) = 1741 - 900 e^{-0,04(n - 30)} , \text{ para } n > 30$$

REEVES (1995) desenvolveu um algoritmo baseado na técnica do Algoritmo Genético para resolver o problema de Programação de Operações *Flow Shop* Permutacional. Nesse trabalho, o autor fez uma comparação entre o Algoritmo

Genético proposto e a técnica *Simulated Annealing*. O Algoritmo Genético se encontra descrito no Capítulo 3.

ISHIBUCHI, MISAKI & TANAKA (1995) desenvolveram dois algoritmos *Simulated Annealing* (SA) modificados para o problema de seqüenciamento *Flow Shop*, a partir do *Simulated Annealing* padrão proposto por OSMAN & POTTS:

- “Algoritmo SA Modificado com a Estratégia da Melhor Mudança”: é semelhante ao *Simulated Annealing* padrão. A única diferença refere-se à probabilidade de aceitação, que é influenciada pela escolha da melhor solução nas K soluções geradas e, portanto, cada geração da solução candidata à solução atual requer a avaliação do *makespan* para K soluções (o número total de iterações é N/K e o número total de avaliações do *makespan* é N). Além disso, a geração da h-ésima solução independe da geração das (h-1) soluções anteriores (algumas soluções podem ser geradas duas vezes ou mais). Nesse algoritmo, a probabilidade de aceitação de cada solução y_h na vizinhança $S(x)$, onde x é a solução atual, é dada pela expressão:

$$G(y_h) = \{h^K - (h-1)^K / S^K\}, \text{ onde:}$$

- S^K : número total de enumerações possíveis das K soluções de $S(x)$ e
- h^K : número total de enumerações possíveis das K soluções de (y_1, y_2, \dots, y_h) .

Se todas as K soluções forem selecionadas de (y_1, y_2, \dots, y_h) , então a solução candidata y é uma das (y_1, y_2, \dots, y_h) soluções. Se $K = 1$, $G(y_h)$ se reduz à mesma probabilidade de aceitação do SA padrão.

- “Algoritmo SA Modificado com a Estratégia da Primeira Mudança”: a solução atual (x) é substituída pela primeira solução que melhore x nas K soluções geradas. Se tal solução não existir, a melhor solução é selecionada como candidata

para substituir a solução atual x , como no “Algoritmo SA Modificado com a Estratégia da Melhor Mudança”.

ZEGORDI, ITOH & ENKAWA (1995) propuseram um método denominado SA-MDJ, onde cada tarefa a ser programada recebe um índice denominado *Move Desirability of Jobs*. Os pares de tarefas a serem trocados são selecionados em função desse índice e o estado de equilíbrio e o critério de parada são definidos no método *Simulated Annealing* proposto. O problema de m máquinas e n tarefas foi aproximado para um problema artificial de duas máquinas de acordo com a regra de JOHNSON. Para gerar o índice MDJ, foi feita uma aproximação baseada no *Slope Index* de PALMER, onde é dada maior prioridade de programação às tarefas cujos tempos de processamento tendem a aumentar de máquina para máquina. Os candidatos à mudança são aqueles que apresentam índices positivos (tarefas com valores de índices máximos são consideradas para troca). Os tempos de processamento para uma tarefa J_i na primeira e na segunda máquina artificial são definidos pelas seguintes expressões:

$$AP_{i1} = \sum_{j=1}^m (m-j+1)t_{ij} \quad e \quad AP_{i2} = \sum_{j=1}^m (j)t_{ij}$$

para: $i = 1, \dots, n$.

O índice MDJ da i -ésima tarefa da seqüência, da mudança para frente e para trás é definido pelas expressões:

$$\bullet D_{[i]}^F = \left\{ \sum_{k=[1]}^{[i-1]} (AP_{[k]1} - AP_{[i]1}) \right\} / AP_{[i]1}, \quad \text{se } AP_{[i]1} \leq AP_{[i]2}$$

$$\bullet D_{[i]}^F = - \left| \sum_{k=[1]}^{[i-1]} (AP_{[k]1} - AP_{[i]1}) \right| / AP_{[i]2}, \quad \text{se } AP_{[i]1} > AP_{[i]2}$$

$$\bullet D_{[i]}^B = \left\{ \sum_{k=[i+1]}^{[n]} (AP_{[k]2} - AP_{[i]2}) \right\} / AP_{[i]2}, \quad \text{se } AP_{[i]2} \leq AP_{[i]1}$$

$$\bullet D_{[i]}^B = - \left| \sum_{k=[i+1]}^{[n]} (AP_{[k]2} - AP_{[i]2}) \right| / AP_{[i]1}, \quad \text{se } AP_{[i]2} > AP_{[i]1}$$

MOTA (1996) apresentou um trabalho que avalia a técnica do Algoritmo Genético aplicada ao problema de Programação de Operações *Flow Shop* Permutacional com o objetivo de minimizar o *makespan*. Foram elaborados diferentes métodos, a partir da utilização de sete operadores de cruzamento distintos, tomados da literatura. São eles: *Partially Matched Crossover*, *Maximal Preventive Crossover*, *Order Crossover*, *Enhance Order Crossover*, *One Point Crossover*, *Cycle Crossover* e *Linear Order Crossover*. Com o objetivo de aumentar a variabilidade dos “cromossomos”, o operador mutação (que apenas troca de lugar duas tarefas aleatoriamente) teve uma frequência de 100% e foi colocado em ação no início e no final do algoritmo. As sub-populações iniciais foram construídas de duas formas: aleatoriamente e utilizando boas soluções iniciais (sementes). Os desempenhos dos diferentes algoritmos foram analisados comparativamente. É importante ressaltar que os problemas resolvidos à partir de boas soluções iniciais (não-aleatórias) tiveram desempenho superior em relação aos problemas resolvidos com sementes iniciais aleatórias. Além disso, os operadores que tiveram melhores desempenhos foram o *Partially Matched Crossover* e o *One Point Crossover*.

PARK & KIM (1998) estudaram o problema de programação *Flow Shop* Permutacional e sugeriram um procedimento para se determinar rapidamente os valores mais apropriados dos parâmetros utilizados no *Simulated Annealing*. O procedimento é baseado no Método Simplex (para programação não-linear).

Este capítulo teve por objetivo caracterizar o problema de Programação de Operações *Flow Shop* Permutacional, bem como apresentar um breve histórico dos trabalhos mais relevantes relatados na literatura. O próximo capítulo apresenta os métodos metaheurísticos Busca Tabu, *Simulated Annealing* e Algoritmo Genético.

CAPÍTULO 3

MÉTODOS METAHEURÍSTICOS

3.1. Aspectos Gerais

O sucesso dos procedimentos heurísticos se justifica pela necessidade atual de se dispor de ferramentas que permitam oferecer soluções rápidas a problemas reais. É importante ressaltar novamente que um método heurístico por si só não garante que a solução encontrada seja a solução ótima do problema. Seu objetivo é encontrar uma solução próxima ao ótimo, em um tempo razoável.

Conforme observado no capítulo anterior, os métodos heurísticos podem ser classificados em construtivos ou melhorativos. Na categoria dos métodos melhorativos, destacam-se os procedimentos de busca em vizinhança. Recentemente, foram desenvolvidos métodos de busca em vizinhança de relativa complexidade (Busca Tabu, *Simulated Annealing* e Algoritmo Genético), que têm sido alvo de interesse da comunidade científica em função de aplicações bem sucedidas reportadas na literatura.

Estas abordagens estão sendo intensivamente empregadas para resolver problemas de otimização combinatorial, como nos casos do problema do caixeiro-viajante, problemas de manutenção, problemas de Programação de Operações em Máquinas, dentre outros.

Uma vez que consistem de princípios de busca geral, tais métodos não podem ser descritos simplesmente como algoritmos, mas como meta-algoritmos. Os meta-

algoritmos são também conhecidos pelos termos metaheurísticos ou heurísticos de busca geral. Estes métodos são desenvolvidos para resolver problemas difíceis de otimização combinatorial, onde os procedimentos heurísticos clássicos não são mais efetivos.

Os métodos metaheurísticos Busca Tabu, *Simulated Annealing* e Algoritmo Genético consistem de procedimentos de busca no espaço de soluções, definidos por estratégias que exploram apropriadamente a topologia de tal espaço.

O sucesso das metaheurísticas se deve a fatores como:

- i) alusão a mecanismos de otimização da natureza (nos casos do Algoritmo Genético e do *Simulated Annealing*);
- ii) aplicabilidade geral da abordagem;
- iii) facilidade de implementação e
- iv) qualidade da solução obtida aliada a um esforço computacional relativamente baixo.

Tais métodos são descritos a seguir, sendo que em função do objetivo deste trabalho, um maior detalhamento é apresentado para o *Simulated Annealing* e o Algoritmo Genético.

3.2. Busca Tabu

Busca Tabu é um procedimento iterativo desenvolvido para resolver problema de otimização combinatorial e que vem sendo bastante utilizado para encontrar a solução ótima ou sub-ótima para problemas de Programação de Operações, otimização de *layout*, dentre outros.

A técnica de Busca Tabu é útil para encontrar uma boa solução, ou possivelmente a solução ótima de problemas do tipo: “Minimizar $c(x)$, sujeito a $x \in X$ ”, onde $c(x)$ é uma função de uma variável discreta x , e X é o conjunto de soluções viáveis. Um “passo” da Busca Tabu inicia com uma solução viável atual $x \in X$, à qual é aplicada uma função $m \in M(x)$ que transforma x em x' , sendo x' uma nova

solução viável ($x' = m(x)$). Esta transformação é denominada uma “mudança” e o conjunto $\{x' : x' = m(x); x, x' \in X; m \in M(x)\}$ é denominado “vizinhança” de x .

Com o objetivo de evitar, o tanto quanto possível, a formação de ciclos nas “mudanças”, um elemento t é associado a m e x . Este elemento define um conjunto de “mudanças” que são Tabu (ou seja, proibidas) e que são guardadas em uma Lista T denominada “Lista Tabu”. Especificamente, o elemento t proíbe a aplicação de m' em x' , o que poderia fazer com que x' se transformasse de volta em x , podendo proibir também outros tipos de “mudanças”. Os elementos da lista T definem todas as mudanças proibidas que não podem ser efetuadas pela solução considerada atual (solução corrente).

Em termos práticos, o tamanho da lista Tabu (T) não pode continuamente crescer, sendo limitado por um parâmetro s , denominado “tamanho da lista Tabu”. Se $|T| = s$, antes de incluir um novo elemento t na lista Tabu, deve-se remover um outro elemento, que é geralmente o mais antigo.

Desta forma, caracteriza-se uma aplicação da técnica de Busca Tabu através dos seguintes componentes:

- (a) O conjunto $M(x)$ de “mudanças” associadas a uma solução viável x (vizinhança);
- (b) O tipo dos elementos do conjunto T , o qual define as “mudanças tabu”, ou seja, proibidas (Lista Tabu);
- (c) O tamanho s do conjunto T (tamanho da Lista Tabu) e
- (d) Uma condição de parada do processo de busca de melhores soluções viáveis x' .

A característica relevante da Busca Tabu refere-se à capacidade de não ficar restrita às soluções ótimas locais, explorando os “vales” do espaço de soluções na tentativa de obtenção da solução ótima global.

Uma descrição completa e detalhada da Técnica de Busca Tabu pode ser encontrada em GLOVER (1989).

3.3. *Simulated Annealing*

3.3.1. Aspectos Gerais

O *Simulated Annealing* tem sua origem na termodinâmica e na metalurgia: quando o aço fundido é resfriado lentamente, tende a se solidificar em uma estrutura de mínima energia. Os estados de energia de um conjunto de partículas podem ser caracterizados como a configuração do material. A probabilidade com que uma partícula esteja em um determinado nível de energia pode ser calculada pela distribuição de Boltzmann. Uma vez que a temperatura do material diminui, a distribuição de Boltzmann tende para a configuração de partícula que tenha a menor energia.

Para eliminar os defeitos de um cristal via resfriamento, deve-se elevar sua temperatura a um nível inferior ao seu ponto de fusão. Em temperaturas elevadas, os defeitos são criados e também eliminados. Quando o equilíbrio térmico é obtido, a taxa de criação de defeitos se iguala à taxa de eliminação. Neste ponto, a temperatura deve ser reduzida, fazendo com que a taxa de eliminação de defeitos exceda a taxa de criação dos mesmos, até que o equilíbrio térmico seja novamente atingido. Repete-se então, o processo.

A idéia de se aplicar o princípio de tratamento térmico (aquecimento e gradativo resfriamento ou *annealing*) aos problemas de otimização partiu de KIRKPATRICK, GELATT & VECCHI (1983), publicada no trabalho *Optimization by Simulated Annealing*. Baseando-se em um algoritmo utilizado na físico-química, eles estabeleceram uma analogia entre conceitos termodinâmicos (como configuração, energia e temperatura) e problemas de otimização combinatorial (solução viável, solução ótima e custo).

O *Simulated Annealing* possui a capacidade de escapar de soluções ótimas locais através da aceitação de soluções que pioram momentaneamente o valor da função-objetivo, sob condições específicas. A diferença fundamental entre a técnica *Simulated Annealing* e as demais técnicas semelhantes (ou seja, melhoria da solução atual através de busca em vizinhança) é que, a solução atual pode ser substituída (de

maneira probabilística) por outra com desempenho inferior quanto à função-objetivo adotada.

Assim como a Busca Tabu, a técnica *Simulated Annealing* tem a capacidade de explorar os “vales” do espaço de soluções na tentativa de obtenção da solução ótima global. Esta técnica vem sendo bastante aplicada na área de Química Analítica e em outros campos da ciência e da engenharia, como por exemplo: problemas de Programação de Operações e problemas de projeto de circuitos.

A utilização do *Simulated Annealing* nos problemas de otimização combinatorial se justifica pelos seguintes fatores:

- i) *Simulated Annealing* é capaz de lidar com funções de custo;
- ii) *Simulated Annealing* é relativamente fácil de ser codificado e implementado, mesmo para os problemas complexos e
- iii) *Simulated Annealing* geralmente encontra uma boa solução.

3.3.2. O Princípio de Funcionamento

De acordo com OSMAN & POTTS (1989), *Simulated Annealing* é um método heurístico que segue os mesmos passos básicos utilizados nos métodos descendentes. Os *descent methods* (métodos descendentes) se caracterizam por tentarem repetidamente melhorar uma solução atual.

A grande diferença entre os métodos descendentes e o *Simulated Annealing* é que este último pode aceitar uma nova solução σ' como a solução atual, mesmo que o valor da função-objetivo $C(\sigma')$ exceda o valor da função-objetivo da solução atual σ .

O *Simulated Annealing* não busca exaustivamente a melhor solução na vizinhança da solução atual. Ele apenas extrai ao acaso uma solução σ' na vizinhança de σ . Se $C(\sigma') \leq C(\sigma)$, então σ' passa a ser a nova solução atual. Porém, se $C(\sigma') > C(\sigma)$, então σ' pode se tornar a solução atual de acordo com uma probabilidade $p(k)$. Tal probabilidade diminui em função do número de iterações do algoritmo. Este método permite, portanto, a mudança de uma solução para outra na

vizinhança com um valor pior da função-objetivo, possibilitando assim um movimento contrário a um eventual ótimo local.

Para testar se σ' é aceita quando $C(\sigma') > C(\sigma)$, um número aleatório R é gerado a partir de uma distribuição uniforme $[0,1]$. Se $R \leq p(k)$, então σ' é aceita como a nova solução; caso contrário, σ se mantém como a solução atual.

A aplicação do *Simulated Annealing* requer tomadas de decisões específicas para cada problema, como a determinação do método de geração da solução inicial, definição da vizinhança, entre outras.

Segundo PARK & KIM (1998), deve-se estabelecer um programa de tratamento térmico que é caracterizado pelos seguintes parâmetros: temperatura inicial (T_1), função de resfriamento ($F(T_k)$), número de iterações (K), condição de parada e probabilidade de aceitação ($p(k)$).

3.3.3. Os Parâmetros Associados ao *Simulated Annealing*

Os valores dos parâmetros que maximizam a performance do *Simulated Annealing* são usualmente obtidos mediante extensivos ensaios. A seleção cuidadosa desses valores contribui de forma decisiva para a boa performance do método heurístico.

⇒ **Temperatura Inicial (T_1):** KIRKPATRICK *et al.* (1983) fixaram um alto valor para T_1 , de modo que a probabilidade inicial de aceitação de mudanças “ruins” (que deterioram momentaneamente o valor da função-objetivo) seja próxima a 1 (um).

O valor da temperatura inicial pode ser determinado de modo que a fração de transições que piorem o valor da função-objetivo em uma iteração, seja igual a um determinado parâmetro P_1 (probabilidade de aceitação inicial). Algumas transições são realizadas e o aumento médio nos valores da função-objetivo (Δ médio) é calculado apenas para as transições ruins. O valor de T_1 é, então, calculado pela expressão:

$$T_1 = - (\Delta \text{ médio}) / \ln P_1$$

Como exemplo, no problema de Programação de Operações *Flow Shop* Permutacional, OSMAN & POTTS (1989) utilizaram a seguinte expressão para determinar a temperatura inicial:

$$T_1 = \sum_{i=1}^n \sum_{j=1}^m p_{ij} / (5mn)$$

- onde:
- p_{ij} = tempo de processamento da tarefa i na máquina j ;
 - n = número de tarefas e
 - m = número de máquinas.

⇒ **Função de Resfriamento ($F(T_k)$):** A temperatura do sistema normalmente se reduz a cada iteração. O parâmetro T (temperatura) diminui de acordo com uma determinada função de resfriamento, até que a condição de parada seja satisfeita. Tipicamente, a temperatura é alta nos estágios iniciais, permitindo assim mudanças de soluções atuais com desempenho inferior. Gradativamente, o valor de T decresce atingindo um valor próximo de zero, de forma que a troca de uma solução atual praticamente só é permitida quando for encontrada uma solução vizinha com desempenho superior. A função de resfriamento pode ser dada, por exemplo, pela seguinte expressão:

$$T_{k+1} = T_k / (1 + \beta T_k)$$

onde:

- T_k = temperatura da k -ésima iteração e
- β é um parâmetro dado pela relação: $\beta = (T_1 - T_K) / [(K - 1) \cdot (T_1 + T_K)]$,

- onde:
- T_1 = temperatura inicial;
 - T_K = temperatura final e
 - K = número de iterações.

⇒ **Número de Iterações (K):** o valor de K pode ser proporcional ao tamanho do problema ou, então, ao número de vizinhos de uma dada solução. Nos problemas de seqüenciamento de máquinas simples, o número de iterações pode ser tomado como o produto do número de tarefas por um parâmetro pré determinado.

No trabalho de OSMAN & POTTS (1989), o número de iterações K (para valores do número de máquinas $m \leq 20$ e número de tarefas $n \leq 100$), foi calculado pela seguinte expressão:

$$K = \max \{3300 \ln n + 7500 \ln m - 18250, 2000\}$$

⇒ **Probabilidade de Aceitação (p(k)):** em analogia à termodinâmica, geralmente toma-se a distribuição baseada em Boltzmann (PIRLOT, 1996), dada por:

$$p(k) = \exp (-\Delta / T_k)$$

- onde:
- $\Delta = C(\sigma') - C(\sigma)$;
 - σ' : solução “candidata”, extraída da vizinhança da solução atual;
 - σ : solução atual e
 - T_k : temperatura no passo k.

⇒ **Critério de Parada:** são admitidos diferentes critérios de parada. A busca pode ser concluída quando o número de iterações atingir um limite pré-determinado. Outra opção é restringir o tempo computacional. A busca pode ser encerrada, ainda, quando a temperatura atingir um valor final pré estabelecido.

Não há regras específicas para se definir os valores mais apropriados para os parâmetros acima relacionados. PIRLOT (1996) afirma que o processo deve se iniciar com uma temperatura inicial T_1 suficientemente alta. Pode-se, por exemplo, selecionar um valor que assegure uma probabilidade de aceitação de mudanças ruins no início. Em relação ao número de iterações K, pode-se tomar um múltiplo do tamanho médio de uma vizinhança.

⇒ **Parâmetros Não-Quantitativos:** a determinação do método de geração da solução inicial e a definição da vizinhança são fundamentais para que o método tenha êxito.

O processo pode ter início a partir de uma solução inicial escolhida aleatoriamente, ou então partindo de uma boa solução inicial, gerada por algum heurístico construtivo.

Tomando-se uma solução viável qualquer do problema de Programação de Operações *Flow Shop* Permutacional, isto é, uma seqüência σ das n tarefas $\{J_1, J_2, \dots, J_n\}$, uma vizinhança de σ seria, por exemplo, o conjunto de todas as seqüências que podem ser obtidas a partir de σ , por uma troca de posições entre as tarefas. A busca na vizinhança pode ser total (todos os vizinhos são examinados) ou parcial (apenas uma parcela pré-determinada ou aleatória do número total de vizinhos é avaliada).

Na Programação *Flow Shop* Permutacional, a vizinhança de uma seqüência σ é normalmente obtida:

i) permutando-se duas tarefas adjacentes, nas posições i e $i + 1$ (*Adjacent Neighbourhood* ou Vizinhança de Adjacência). O tamanho da vizinhança para uma seqüência σ com n tarefas é: $(n-1)$;

ii) permutando-se as tarefas nas posições i e k , sendo $i \neq k$ (*Interchange Neighbourhood* ou Vizinhança de Permuta). Para uma seqüência σ , o tamanho dessa vizinhança é: $n \cdot (n-1) / 2$ ou

iii) removendo-se a tarefa da posição k e colocando-a na posição i (*Shift Neighbourhood* ou Vizinhança de Inserção). O tamanho dessa vizinhança é: $(n-1)^2$.

3.3.4. Estrutura Básica do *Simulated Annealing*

- Seja:
- X = conjunto das soluções viáveis x ;
 - $F(x)$ = valor da função-objetivo associado à solução x ;
 - x^* = a melhor solução já obtida;
 - F^* = melhor valor da função-objetivo e

→ $p(k)$ = probabilidade de aceitação.

INÍCIO

- ➔ Selecione uma solução inicial x_1 pertencente ao conjunto X . Inicialize o melhor valor F^* e a solução correspondente x^* .

$$F^* \leftarrow F(x_1)$$

$$x^* \leftarrow x_1$$

- ➔ Enquanto a condição de parada não for satisfeita:

- Extraia aleatoriamente x a partir da vizinhança $V(x_k)$ da solução atual x_k
- Se $F(x) \leq F(x_k)$, então $x_{k+1} \leftarrow x$
- Senão, tome ao acaso um valor R do intervalo $[0,1]$. Se $R \leq p(k)$, então:
$$x_{k+1} \leftarrow x, \text{ caso contrário } x_{k+1} \leftarrow x_k$$
- Se $F(x_{k+1}) < F^*$, então $F^* \leftarrow F(x_{k+1})$ e $x^* \leftarrow x_{k+1}$

FINAL.

3.4. Algoritmo Genético

3.4.1. Aspectos Gerais

O Algoritmo Genético pode ser descrito como um mecanismo que imita a evolução genética das espécies, pois considera como fatores evolutivos: as mutações, a recombinação gênica, as migrações, o tamanho da população, a seleção natural,

entre outros. São métodos generalizados de busca e otimização que simulam os processos naturais da evolução, aplicando a idéia Darwiniana de seleção. Todos estes fatores são simulados na forma de sistemas artificiais, como programas de computador.

Nos anos 50 e 60, muitos biólogos passaram a desenvolver simulações computacionais de sistemas genéticos. Contudo, as primeiras pesquisas mais sérias começaram a ser desenvolvidas por HOLLAND (1975), autor do livro intitulado *Adaptation in Natural and Artificial Systems*, considerado a obra clássica sobre Algoritmo Genético. O campo de aplicação desses algoritmos vem se desenvolvendo devido principalmente às inovações computacionais realizadas nos anos 80.

Além de ser uma estratégia diferenciada, pois se baseia na evolução biológica, os Algoritmos Genéticos são capazes de identificar e explorar fatores ambientais (do espaço de soluções) e convergir para soluções ótimas ou aproximadamente ótimas em níveis globais e em uma grande variedade de problemas.

De acordo com GOLDBERG (1989), quatro características distinguem o Algoritmo Genético:

(1) Trabalha com uma codificação do conjunto de parâmetros, e não com os próprios parâmetros. O Algoritmo Genético requer que os parâmetros utilizados sejam codificados sob a forma de séries finitas;

(2) Trabalha a partir de uma população de soluções viáveis, e não com soluções isoladas. Isto faz com que a probabilidade de se localizar uma solução ótima local no espaço de busca seja reduzida em relação aos métodos que trabalham com soluções isoladas;

(3) Utiliza regras probabilísticas de mudança de soluções, e não determinísticas. Isto não significa que o Algoritmo Genético seja um método de busca aleatório. Ele emprega uma estratégia de busca aleatória, o que não implica, necessariamente, em uma busca não direcionada, uma vez que ele explora informações históricas para encontrar novos pontos de busca onde são esperados desempenhos melhores e

(4) Utiliza informações da função-objetivo, geralmente não-diferenciáveis e não necessita de outro conhecimento auxiliar. Alguns métodos utilizam o cálculo de derivadas, outros precisam ter acesso à maioria dos parâmetros do problema. O Algoritmo Genético realiza sua busca por melhores soluções utilizando somente os valores das funções-objetivo associados a cada solução individual.

O Algoritmo Genético parte da premissa de que problemas complexos podem ser resolvidos simulando-se a evolução natural através de um algoritmo computacional que manipula séries binárias denominadas cromossomos. Apesar de parecer simplista, este método apresenta complexidade suficiente para fornecer mecanismos de busca poderosos e eficientes.

Dentre as diversas áreas de aplicação dos Algoritmos Genéticos, destacam-se: problemas de Programação de Operações em Máquinas; biologia molecular e físico-química; engenharia de construções; buscas em base de dados; redes neurais; dentre outras.

3.4.2. O Princípio de Funcionamento

Basicamente, o Algoritmo Genético trabalha da seguinte maneira: inicialmente, é gerada uma população formada por um conjunto de indivíduos (são possíveis soluções do problema). O princípio de funcionamento do Algoritmo Genético é que, após inúmeras gerações (iterações), o conjunto inicial de indivíduos gere novos indivíduos mais aptos, utilizando um critério de seleção. Este critério procura escolher indivíduos com maiores notas de aptidão: cada indivíduo recebe um índice, refletindo sua habilidade de adaptação ao ambiente, ou seja, pontos nos quais a função a ser minimizada tem valores relativamente baixos. Apenas os membros mais adaptados são mantidos pela seleção.

É necessário também um conjunto de operadores para que, a partir de uma determinada população, sejam geradas populações sucessivas que melhorem sua aptidão com o tempo. Os indivíduos mais aptos sofrem modificações em suas características fundamentais, originando descendentes para a próxima geração. Para prevenir o desaparecimento dos melhores indivíduos da população (pela manipulação

dos operadores), eles devem ser colocados automaticamente na próxima geração, através de uma reprodução elitista. Este ciclo se repete por um determinado número de vezes.

Na terminologia do Algoritmo Genético, um cromossomo representa uma solução para o problema em estudo, sendo constituído de um conjunto de genes cada qual representando uma característica específica do cromossomo. O valor associado à característica de cada gene é denominado alelo. Uma população é constituída de um conjunto de cromossomos, sendo refeita (atualizada, renovada) após cada geração. Os cromossomos representam indivíduos que são levados ao longo de várias gerações, de forma similar aos problemas naturais, evoluindo de acordo com os princípios de seleção natural e sobrevivência dos mais aptos.

Um elemento fundamental do Algoritmo Genético é o operador de cruzamento, o qual combina dois cromossomos para gerar cromossomos “filhos” que mantêm certas características básicas dos cromossomos pais. Além desse operador, também é geralmente utilizado o operador de mutação, o qual introduz aleatoriamente transformações ou novas informações em uma população.

Na aplicação do Algoritmo Genético para solução do problema de Programação de Operações *Flow Shop* Permutacional, um cromossomo representa uma possível seqüência das tarefas e os genes representam as tarefas.

O presente trabalho utiliza um Algoritmo Genético simples, devido à sua facilidade de compreensão, implementação e eficiência/eficácia, restringindo-se a copiar cromossomos (soluções do problema) e trocar suas partes.

A força do Algoritmo Genético baseia-se na hipótese de que a melhor solução se encontra em regiões do espaço de soluções viáveis que contêm uma grande proporção relativa de boas soluções e que tais regiões podem ser identificadas através de uma amostragem minuciosa do espaço de soluções.

3.4.3. O Algoritmo Genético Simples

Ele é denominado Algoritmo Genético simples quando utiliza somente os operadores genéticos reprodução, cruzamento e mutação. Os operadores genéticos transformam a população através de sucessivas gerações, estendendo a busca até se

chegar a um resultado satisfatório. Eles são necessários para que a população se diversifique e mantenha as características de adaptação adquiridas pelas gerações anteriores. Uma breve descrição desses operadores é apresentada a seguir.

➡ **Operador Reprodução:** reprodução é o processo pelo qual cromossomos são escolhidos a partir de uma população, para reciclar e melhorar esta população. Para se iniciar a reprodução, deve-se ter uma população inicial e cada indivíduo desta população deve ser avaliado por uma função-objetivo. A reprodução faz o papel da seleção natural, pois para cada cromossomo (uma solução viável do problema) há um valor associado.

Para que a variabilidade dos indivíduos aumente, pode-se partir de duas subpopulações. Em relação ao problema de Programação de Operações, toma-se como pais as seqüências das tarefas com os menores valores da função-objetivo. Através da seleção, determina-se quais indivíduos se reproduzirão, gerando um número determinado de descendentes para a próxima geração.

➡ **Operador de Cruzamento:** na natureza, cruzamento é o fenômeno em que dois pais trocam partes de seus cromossomos para gerar filhos. No Algoritmo Genético, o operador de cruzamento recombina o material genético codificado em dois cromossomos-pais para produzir dois descendentes. O cruzamento é, portanto, responsável pela recombinação das características dos pais, permitindo que as próximas gerações herdem tais características.

Ele é executado após a reprodução e pode ser aplicado em duas etapas: inicialmente, cada pai escolhido na reprodução é “cortado” em determinadas posições e, em seguida, cada par de pais tem as partes trocadas entre si.

Existem diferentes operadores de cruzamento. Um dos mais empregados e de mais fácil aplicação é o **cruzamento de um corte** (*one point crossover*), onde um ponto de cruzamento é escolhido e, a partir deste ponto, as informações dos pais são trocadas.

A Figura 2 ilustra o funcionamento deste operador.

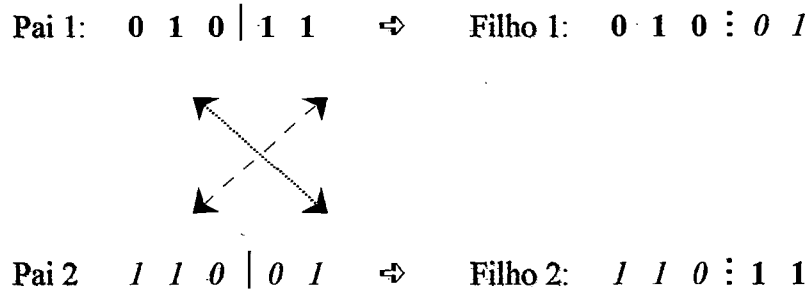


FIGURA 2: Exemplo de cruzamento de um corte

➡ **Operador Mutação:** é responsável por introduzir diversificação genética na população, alterando ao acaso, um ou mais componentes da estrutura genética, gerando assim, novos elementos para a população. A mutação contorna o problema de mínimos locais, pois a direção de busca é alterada. Geralmente, esse operador é aplicado ocasionalmente e com uma taxa pequena. Ele é necessário porque os operadores anteriores podem fazer com que algum material genético útil se perca, isto é, alguma solução importante. Com a utilização da mutação, isto pode ser corrigido.

Um exemplo do seu mecanismo de funcionamento é apresentado na Figura 3.

ANTES da Mutação:

1 1 0 0 1 0



DEPOIS da Mutação:

1 0 0 0 1 0

FIGURA 3: Mecanismo de funcionamento do operador mutação

3.4.4. Estrutura Básica do Algoritmo Genético Simples

INÍCIO

$i \leftarrow 0$.

Inicializar uma população $G(i)$ com N soluções viáveis.

REPETIR

- ➡ Selecionar os pais de $G(i)$ observando o valor da função-objetivo (Reprodução).
- ➡ Aplicar o operador de cruzamento nos pais.
- ➡ Aplicar, se for o caso, o operador mutação e obter a geração $G(i + 1)$.

$i \leftarrow i + 1$.

ATÉ UMA CONDIÇÃO DE PARADA

FIM.

3.5. Métodos Metaheurísticos Híbridos

Uma idéia interessante que tem recebido gradativa atenção refere-se ao desenvolvimento de métodos metaheurísticos híbridos utilizando as metaheurísticas Busca Tabu (BT), *Simulated Annealing* (SA) e Algoritmo Genético (AG). Assim, o objetivo é combinar as técnicas BT, SA e AG, preservando suas características de ação “inteligente”, de tal forma que o procedimento resultante seja mais eficaz do que qualquer um dos seus componentes isoladamente.

Na literatura já foram reportados alguns trabalhos relatando experimentações com variantes ou combinações desses três métodos, desenvolvidos para resolverem diversos problemas. Alguns dos métodos são apresentados a seguir:

KURODA & KAWADA (1994) pesquisaram a melhoria na eficiência computacional das análises de redes de filas inversas (*Inverse Queueing Network Analysis* ou IQNA). Os autores desenvolveram um método de otimização que usa um modelo de rede de fila aliado a um algoritmo de busca local. IQNA é aplicada em problemas de controle de *inputs* em ambiente *Job Shop* com as seguintes propriedades:

múltiplos tipos de produtos manufaturados repetida e simultaneamente; roteiros e tempos de processamento diferentes; grande número de operações e lotes de produção pequenos. O problema de controle de *inputs* se caracteriza pela decisão do nível ótimo de trabalho em processo que produz a quantidade próxima à requerida e minimiza o tempo de permanência (*residence*) na fábrica. O tempo de resolução das análises de rede de filas aumenta polinomialmente de acordo com a escala do modelo. A melhor estratégia para melhorar sua eficiência computacional é reduzir o número de análises realizadas.

Segundo os autores, o *Simulated Annealing* pode ser utilizado na estrutura do IQNA pois ele trata as funções de avaliação múltipla, de forma fácil e apropriada. Porém, quando existe uma direção exata para estas funções, o *Simulated Annealing* negligencia tal direção. Devido a estas limitações, KURODA & KAWADA apresentaram um algoritmo *Simulated Annealing*-Busca Tabu com as seguintes características: a Busca Tabu é utilizada se todas as funções de avaliação forem sucessivamente melhoradas e, quando esta melhoria não for mais possível, muda-se para o *Simulated Annealing*. Dependendo dos valores iniciais do nível de operação em processo de um produto i , o *Simulated Annealing* é utilizado inicialmente e, então, muda-se para Busca Tabu. As temperaturas são reduzidas de acordo com o número de análises de redes de filas. Para evitar a otimização local, o algoritmo foi modificado para aumentar a temperatura temporariamente quando ele se depara com a situação em que muda-se da Busca Tabu para o *Simulated Annealing* após um número pequeno de transições sucessivas.

KIM, NARA & GEN (1994) estudaram o problema de programação de manutenção de unidades térmicas em um sistema de energia. Este problema visa determinar o programa de manutenção de unidades térmicas que busca minimizar o

uso de combustível e o custo de manutenção, além de nivelar as reservas durante um período de tempo completo, sob certas restrições. Os autores propuseram um método metaheurístico híbrido que combina o Algoritmo Genético e o *Simulated Annealing*. O método proposto utiliza a probabilidade de aceitação do *Simulated Annealing* como a probabilidade de sobrevivência de descendentes fracos no processo de evolução do Algoritmo Genético e inibe a ocorrência de cromossomos mortos pela técnica de *decoding/encoding* (um cromossomo representa uma programação de manutenção da unidade completa).

Inicialmente, são executados cruzamento e mutação em cada cromossomo (o processo de seleção é executado de forma aleatória). Dois cromossomos pais são selecionados e produzem dois novos cromossomos filhos. Entre os quatro candidatos para a próxima geração, dois são selecionados em função de suas funções-objetivo. Quando o valor do filho é menor que o valor do pai, ele tem chances adicionais de “sobrevivência” de acordo com certa probabilidade. Isto previne o fenômeno de prematuridade e oferece novos modelos.

O Algoritmo Genético proposto melhora a convergência adotando a probabilidade $\exp(-\Delta E/T^k)$, como probabilidade de sobrevivência dos descendentes fracos, onde ΔE representa a alteração da função objetivo e T^k a temperatura corrente do *Simulated Annealing*. Os autores observaram que nos últimos estágios de busca, dificilmente o cromossomo cujo valor da função-objetivo seja pior é aceito como novo descendente, o que seria esperado em função da característica do método *Simulated Annealing*.

GLOVER, KELLY & LAGUNA (1995) estudaram as metaheurísticas Algoritmo Genético e Busca Tabu, e propuseram princípios para a construção de procedimentos híbridos, a partir da combinação dos elementos-chave e das características complementares de cada método. Segundo os autores, a classificação de uma mudança como sendo tabu (na Busca Tabu) depende da frequência com que determinada mudança ou solução tenha participado da geração de soluções passadas. Para prevenir a busca através de combinações de trocas realizadas recentemente, são classificadas como “tabu” todas as trocas compostas pelos pares mais recentes (*recency*). Frequência e *recency* são chamados de atributos. Quando uma mudança

tabu resulta em uma solução melhor do que aquelas visitadas até o momento, a classificação tabu pode ser ignorada - é o critério da aspiração. Intensificação e diversificação são as bases da memória de longo prazo da Busca Tabu. A estratégia de intensificação em Busca Tabu tem uma contrapartida no Algoritmo Genético: é o método *scatter search*, que foi projetado para operar sobre um conjunto de pontos de referência, que se constituem de boas soluções obtidas previamente. Combinações lineares desses pontos são sistematicamente geradas para criar novos pontos. Os vetores resultantes das combinações lineares dos pontos escolhidos servem de *inputs* para os processos heurísticos, que os transformam em resultados melhores, completando o ciclo. Estes resultados, por sua vez, fornecerão um novo conjunto de pontos de referência, e o processo se reinicia.

Scatter search se assemelha ao Algoritmo Genético, uma vez que ambos são abordagens baseadas em população e criam novos elementos combinando os elementos já existentes. A grande diferença entre o Algoritmo Genético e *scatter search* é que este último não apresenta o recurso da aleatoriedade, ainda que ele possa ser utilizado. *Scatter search* escolhe somente as boas soluções como base para a geração de combinações, enquanto o Algoritmo Genético admite soluções de todos os tipos para serem combinadas. Concluindo, os autores apresentam princípios para a hibridização:

(i) os alelos do Algoritmo Genético (valores das variáveis em um vetor de solução binária) podem ser comparados aos atributos da Busca Tabu. A introdução de memória no Algoritmo Genético para fazer um acompanhamento dos alelos pode criar condições de hibridização com Busca Tabu;

(ii) as representações binárias (no Algoritmo Genético) apresentam certas limitações que podem ser amenizadas. As restrições tabu e o critério de aspiração lidam com aspectos binários de complementariedade sem referências explícitas ao vetor "x" (0-1). A adoção de uma orientação similar pode beneficiar o Algoritmo Genético no procedimento relativo à representação genética;

(iii) os Algoritmos Genéticos modernos que dependem de outros métodos para explorar estruturas (como *Parallel Genetic Algorithms* e *Genetic Local Search*) são, na verdade, métodos híbridos e podem utilizar a Busca Tabu para esta exploração e

(iv) estratégias de intensificação e diversificação na Busca Tabu se baseiam no modo como as informações podem ser extraídas das boas soluções para auxiliar na descoberta de soluções adicionais. O Algoritmo Genético pode ajudar nesta questão, aproximando as soluções e trocando seus componentes.

ROACH & NAGI (1996) analisaram o problema de montagem em múltiplos níveis em uma instalação de manufatura que opera em ambiente *Just-In-Time*. Considerou-se uma instalação constituída de m centros de trabalho, cada um composto de f_i máquinas idênticas, um conjunto de montagens finais p_f e um conjunto de todas as peças manufaturadas $p_m \supset p_f$. Cada peça manufaturada tem uma seqüência única de operações e cada operação requer processamento em um dos m centros de trabalho. O programa de produção consiste de uma lista de produtos acabados, juntamente com as quantidades associadas e *due-dates*. O objetivo é determinar a programação das operações que minimiza o *lead time* de produção. Os autores desenvolveram um algoritmo híbrido que executa o Algoritmo Genético e o *Simulated Annealing* em duas fases que se alternam até a convergência.

Na primeira fase, o Algoritmo Genético gera as soluções iniciais (apenas uma vez) aleatoriamente ou usando heurísticos conhecidos. Estas soluções são modificadas utilizando cruzamento e seleção para produzir novas e melhores soluções. Após dez gerações, as soluções são melhoradas pelo *Simulated Annealing* (segunda fase). Quando o *Simulated Annealing* finaliza suas operações, o Algoritmo Genético continua a criar novas gerações. A mudança Algoritmo Genético/*Simulated Annealing* prossegue até que não haja aperfeiçoamento na melhor solução após cada metaheurística ter completado suas operações. O cromossomo foi representado como um encadeamento de operações em um centro de trabalho; os cromossomos “pais” foram escolhidos via seleção *Roulette Wheel*. O operador de cruzamento utilizado foi o *One Point Crossover* e o operador mutação não foi considerado.

Na segunda fase, a implementação do *Simulated Annealing* baseia-se no conceito clássico da geração de uma seqüência de soluções para uma temperatura particular T . Inicialmente a temperatura é alta e é, então, reduzida em passos. As iterações iniciais do Algoritmo Genético são seguidas pelo *Simulated Annealing* inicializado a alta temperatura. Esta, por sua vez, é progressivamente diminuída. A

programação, de acordo com a seqüência pré-ordenada, se inicia pela operação final, mantendo-se este processo (de trás para frente) até que restrições venham impedir a programação de novas operações no centro de trabalho. Muda-se, então, para um novo centro e programam-se as operações até que restrições impeçam a programação das demais operações. Mantém-se este processo até que todas as operações sejam programadas.

PIRLOT (1996) descreveu em seu trabalho as versões básicas do Algoritmo Genético, Busca Tabu e *Simulated Annealing*. Em relação à hibridização, o autor destaca as seguintes técnicas:

(i) Busca Tabu e *Simulated Annealing* - no algoritmo *Simulated Annealing*, a seleção aleatória de uma solução na vizinhança $V(x_n)$ pode ser substituída pela seleção da melhor solução em uma vizinhança gerada aleatoriamente $V'(x_n) \subseteq V(x_n)$. Uma outra variante do *Simulated Annealing* utiliza alguns conceitos da terminologia da Busca Tabu, através da modificação do programa usual de resfriamento: quando a busca torna-se ineficiente, o sistema é aquecido novamente e a busca se reinicia a uma temperatura maior que, por sua vez, é menor que a temperatura inicial e diminui de ciclo para ciclo (semelhante à aplicação do princípio da “oscilação estratégica”);

(ii) Busca Tabu e Algoritmo Genético - MOSCATO (1993) *apud* PIRLOT (1996) propôs uma nova versão da Busca Tabu onde uma população de soluções é desenvolvida em uma alternância de fases individuais e coletivas: durante as fases coletivas, pares de soluções são misturados utilizando-se o operador de cruzamento.

Com relação ao **problema de Programação de Operações *Flow Shop***, são encontrados na literatura os trabalhos relatados a seguir.

MURATA, ISHIBUCHI & TANAKA (1996) examinaram duas hibridizações do Algoritmo Genético com outros algoritmos de busca. Os autores apresentaram os seguintes algoritmos híbridos:

(i) *Genetic Local Search Algorithm (GLSA)* - semelhante ao Algoritmo Genético, exceto pela introdução de um passo extra entre o passo inicialização (geração da população inicial) e o passo seleção: este passo extra baseia-se na

aplicação de Busca Local às soluções na população atual até que a condição de parada seja satisfeita. Caso a Busca Local seja completada para todas as soluções, então as soluções ótimas locais obtidas tornam-se a população atual. Os demais passos do Algoritmo Genético são seguidos normalmente;

(ii) *Genetic Simulated Annealing (GSA)* - utiliza o *Simulated Annealing* ao invés da Busca Local, ou seja, o *GSA* é obtido apenas modificando-se o passo extra do *GLSA*, isto é, troca-se Busca Local pelo *Simulated Annealing*. Os demais passos deste algoritmo são idênticos aos do Algoritmo Genético. Os autores utilizaram temperatura constante para cada solução na população atual com o objetivo de evitar a deterioração da solução atual durante o estado inicial de resfriamento a altas temperaturas.

DÍAZ (1996) apresentou um método que procura minimizar o custo das tarefas entregues em atraso em problemas de Programação de Operações *Flow Shop* proporcional (quando uma tarefa tem o tempo de processamento longo em uma máquina, então as demais tarefas apresentarão tendência similar em relação à mesma máquina). Para que *Simulated Annealing* e Busca Tabu possam ser aplicados, é necessário ter como ponto de partida uma solução inicial com qualidade. Neste caso, o autor observou que o Algoritmo de Ow (OW, 1985) fornecia um bom ponto inicial S_{OW} a um baixo custo computacional. O autor aplicou o *Simulated Annealing* à solução inicial S_{OW} , usando parâmetros previamente determinados, obtendo uma nova solução S_{SA} que, por sua vez, foi utilizada como solução inicial na implementação da Busca Tabu. Os resultados dos experimentos efetuados mostraram que, enquanto o custo computacional do Algoritmo de Ow é insignificante, o do *Simulated Annealing* aumenta lentamente com o número de tarefas e o da Busca Tabu cresce exponencialmente.

A literatura relativa ao desenvolvimento e aplicação de métodos metaheurísticos híbridos ao problema de Programação de Operações *Flow Shop* não é muito vasta. Alguns trabalhos foram realizados e acredita-se que outros estão sendo desenvolvidos. Contudo, este campo ainda apresenta um grande potencial de desenvolvimento.

A partir da revisão da literatura, considera-se importante reiterar o objetivo deste trabalho:

O presente trabalho tem como objetivo desenvolver um método metaheurístico híbrido para o Problema de Programação de Operações *Flow Shop* Permutacional buscando a minimização do makespan, utilizando-se as metaheurísticas Algoritmo Genético e *Simulated Annealing*.

O método híbrido proposto, bem como os métodos *Simulated Annealing* puro e Algoritmo Genético puro que foram utilizados na sua concepção, são apresentados no próximo capítulo.

CAPÍTULO 4

O MÉTODO METAHEURÍSTICO HÍBRIDO PROPOSTO

4.1. Aspectos Gerais

Partindo-se do estudo realizado a respeito dos métodos metaheurísticos Algoritmo Genético e *Simulated Annealing* puros, e das hibridizações reportadas na literatura, elaborou-se um método metaheurístico híbrido cuja base é composta pela metaheurística Algoritmo Genético simples, associada a um método metaheurístico responsável pela geração de diversificação genética - o *Simulated Annealing*.

O método híbrido – denominado *HBGASA* - proposto no presente trabalho tem como ponto de partida duas soluções iniciais geradas por dois métodos heurísticos construtivos de comprovada eficiência/eficácia. O seu objetivo é melhorar estas soluções, através de um procedimento que combina as abordagens metaheurísticas Algoritmo Genético e *Simulated Annealing*, obtendo uma solução final de melhor qualidade.

Conforme já mencionado, a aplicação do *Simulated Annealing* para a solução de problemas de otimização requer a tomada de decisões específicas para cada problema, tais como a determinação de métodos de geração da solução inicial; a definição da vizinhança a ser considerada; a condição de parada, dentre outras.

Em relação ao Algoritmo Genético, a especificação do “cenário” em que o mesmo deve atuar também é de fundamental importância para a maximização do seu desempenho, como por exemplo: a determinação da forma de construção das sub-populações iniciais; a escolha dos operadores de cruzamento e mutação que serão empregados; a frequência de utilização do operador mutação, dentre outros.

O dimensionamento dos parâmetros e elementos utilizados no método híbrido *HBGASA* é, portanto, decisivo para que seu desempenho seja maximizado. Uma apresentação dos elementos é fornecida a seguir.

4.2. Parâmetros Quantitativos e Não-Quantitativos do Método Híbrido *HBGASA*

4.2.1. Soluções Iniciais

As soluções iniciais são fornecidas pelo método heurístico construtivo NEH (NAWAZ, ENSCORE & HAM, 1983) e pela solução inicial do método FSHOPH (MOCCELLIN, 1995). No método *Simulated Annealing* puro que é comparado com o híbrido *HBGASA*, a solução inicial é a melhor dentre as soluções iniciais do *HBGASA*.

4.2.2. Vizinhança

No presente trabalho, foi utilizado o tipo de Vizinhança de Inserção (*Shift Neighbourhood*), onde uma seqüência vizinha é obtida da seqüência atual removendo-se uma tarefa de sua posição e inserindo-a em uma outra posição.

A Vizinhança de Inserção é capaz de gerar maior variedade de vizinhos, quando comparada à Vizinhança de Adjacência e à Vizinhança de Permuta (vide Capítulo 3), justificando assim, a sua utilização.

4.2.3. Formas de Busca na Vizinhança

Foi empregada a **busca parcial aleatória**, que consiste no exame parcial da vizinhança, ou seja, apenas uma parcela $Nvp(S)$ do número total de vizinhos é avaliada, a qual é gerada aleatoriamente.

A parcela da vizinhança, baseada no trabalho desenvolvido por NAGANO (1995), é dada pelas seguintes expressões:

$$Nvp(S) = p (n - 1)^2, \text{ para } n \leq 30$$

$$Nvp(S) = p [1741 - 900 \exp(-0,04(n - 30))] , \text{ para } n > 30$$

onde p é um parâmetro a ser obtido experimentalmente no conjunto {0,05; 0,10; 0,15; 0,20; 0,25; 0,30; 0,35; 0,40; 0,45; 0,50; 0,55; 0,60; 0,65; 0,70; 0,75; 0,80; 0,85; 0,90; 0,95; 1,00} e n é o número de tarefas a serem programadas.

4.2.4. Condição de Parada

O método híbrido encerra a busca por melhores soluções após um número $TNbS(m,n)$ pré-determinado de seqüências avaliadas de acordo com a Tabela 1. Tais números foram obtidos por experimentação prévia: para cada classe de problema (número de máquinas m , número de tarefas n), o valor $TNbS(m,n)$ corresponde ao número médio de seqüências avaliadas utilizando-se o método heurístico de Busca Tabu FShop.TS5 (MOCCELLIN & NAGANO, 1998).



TABELA 1: Condição de Parada

Número de (máquinas, tarefas)	TNbS(m,n)	Número de (máquinas, tarefas)	TNbS(m,n)	Número de (máquinas, tarefas)	TNbS(m,n)
(4, 20)	9693	(7, 20)	10812	(10, 20)	16750
(4, 30)	43732	(7, 30)	59666	(10, 30)	73503
(4, 40)	77742	(7, 40)	162358	(10, 40)	125728
(4, 50)	105623	(7, 50)	165856	(10, 50)	154223
(4, 60)	133502	(7, 60)	178017	(10, 60)	166257
(4, 70)	137452	(7, 70)	135330	(10, 70)	173862
(4, 80)	166050	(7, 80)	107244	(10, 80)	189378
(4, 90)	155455	(7, 90)	128733	(10, 90)	189406
(4, 100)	194680	(7, 100)	108136	(10, 100)	190884

Convém ressaltar que os dois métodos *puros* (*algoritmo genético e Simulated Annealing*), que são comparados com o *híbrido*, utilizam a mesma condição de parada.

4.2.5. Operador de Reprodução

A reprodução é efetuada sobre a população constituída pelas duas seqüências “pais” e duas seqüências “filhos”. As duas seqüências que apresentarem as menores durações totais de programação são selecionadas para a aplicação do *operador de cruzamento*.

4.2.6. Operador de Cruzamento

Foram selecionados dois tipos de operadores de cruzamento, a partir dos resultados reportados por MOTA (1996). Tais operadores mostraram-se os mais eficazes dentre os operadores de cruzamento analisados no referido trabalho. Eles são descritos a seguir.

① *One Point Crossover* ou Operador de Cruzamento de Um Corte

- PASSO 1: Escolha um ponto para se dividir cada pai, em duas subsequências;
- PASSO 2: A solução filho é gerada através da união de duas subsequências, uma de cada pai, na sua posição absoluta;
- PASSO 3: Para evitar a inviabilidade da solução filho, trocam-se as tarefas duplicadas por tarefas que faltam para completar a solução filho.

Exemplo:

- PASSO 1:

A: 1 2 3 4 5 6 7 | 8 9 10
 B: 4 9 5 2 3 8 10 | 1 7 6

- PASSO 2:

A': 1 2 3 4 5 ~~6~~ ~~7~~ : 1 7 6
 B': 4 ~~9~~ 5 2 3 8 10 : 8 9 10

- PASSO 3:

A': 8 2 3 4 5 10 9 1 7 6
 B': 4 7 5 2 3 1 6 8 9 10

② *Partially Matched Crossover* ou Operador de Cruzamento PMX

- PASSO 1: Escolha um intervalo para ser trocado;
- PASSO 2: Crie um mapa do intervalo selecionado;
- PASSO 3: Permute os dois intervalos;
- PASSO 4: Use o mapa para eventualmente alterar as soluções filhos e torná-las viáveis.

Exemplo:

$$\leq n/2$$

A: 9 8 4 5 6 7 1 3 2 10
 B: 8 7 1 2 3 10 9 5 4 6

- PASSO 1: Escolha um intervalo (neste exemplo, posições 4 a 6)

- PASSO 2:

A	B
5	2
6	3
7	10

- PASSO 3:

A': 9 8 4 2 3 10 1 3 2 10
 B': 8 7 1 5 6 7 9 5 4 6

- PASSO 4:

A': 9 8 4 2 3 10 1 6 5 7
 B': 8 10 1 5 6 7 9 2 4 3

4.2.7. Operador de Mutação

Este operador é necessário para a introdução e a manutenção da diversidade genética da população. Apesar da sua relativa importância, ele *não é utilizado no método híbrido HBGASA*, uma vez que nesse método tal função é realizada pelo procedimento *Simulated Annealing*.

Porém, para o Algoritmo Genético puro (a ser comparado com o método híbrido) foram desenvolvidos 3 operadores de mutação, a saber:

i) operador *mutação 1*: consiste em escolher aleatoriamente uma tarefa e trocá-la de posição com a sua subsequente;

ii) operador *mutação 2*: troca aleatoriamente as posições de duas tarefas quaisquer da seqüência e

iii) operador *mutação 3*: escolhe ao acaso um ponto para se dividir a seqüência em duas subseqüências, as quais têm suas posições relativas invertidas.

Pode-se notar que os operadores de mutação propostos têm graus de “perturbação” crescentes. O operador *mutação 1* é aplicado toda vez que não houver melhoria na melhor seqüência até então obtida, após $0,01TNbS(m,n)$ seqüências sucessivas geradas e avaliadas. O *mutação 2* é utilizado se não houver melhoria após $0,027TNbS(m,n)$ seqüências sucessivas. O operador *mutação 3*, que causa uma maior modificação na seqüência gerada é aplicado após $0,10TNbS(m,n)$ seqüências sucessivas sem melhoria quanto à melhor solução até o momento obtida. Os coeficientes 0,01; 0,027 e 0,10 foram estabelecidos em função do “grau de perturbação” de cada operador e também de forma que não haja a possibilidade de aplicação simultânea de dois ou dos três operadores.

4.2.8. Temperatura Inicial T_1

A temperatura inicial é um dos parâmetros sensíveis e experimentais do método híbrido *HBGASA*. Com o objetivo de obter o melhor valor para tal parâmetro, foi estabelecido um conjunto representativo dado por {20, 30, 45, 60, 70}.

4.2.9. Função de Resfriamento $F(T_k)$

No procedimento *Simulated Annealing*, após cada iteração ocorre uma diminuição da temperatura conforme a seguinte expressão:

$$T_{k+1} = T_k / (1 + \beta T_k)$$

onde:

- T_k = temperatura da k-ésima iteração;

- β é um parâmetro dado pela relação: $\beta = (T_1 - T_K) / [(K - 1) \cdot (T_1 + T_K)]$, sendo:
 - T_1 = temperatura inicial;
 - T_K = temperatura final e
 - K = número de iterações.

O número de iterações K depende da estrutura do método, puro ou híbrido, sendo uma função da **condição de parada**. A temperatura final é adotada igual a 1 (um).

4.2.10. Probabilidade de Aceitação $p(k)$

A probabilidade de aceitação de um “movimento” no procedimento *Simulated Annealing* é calculada pela distribuição de Boltzmann, ou seja :

$$p(k) = \exp(-\Delta / T_k)$$

onde:

- $\Delta = C(\sigma') - C(\sigma)$;
- σ' : solução “candidata”, extraída da vizinhança da solução atual (corrente);
- σ : solução atual e
- T_k : temperatura na iteração k .

É importante salientar também, que o método híbrido proposto se diferencia dos métodos híbridos Algoritmo Genético - *Simulated Annealing* encontrados na literatura:

Conforme observado no Capítulo 4, ROACH & NAGI (1996) propuseram um algoritmo híbrido para o problema de montagem em múltiplos níveis (ambiente *Just In Time*), que executa o Algoritmo Genético e o *Simulated Annealing* em duas

fases que se alternam: na sua primeira, o Algoritmo Genético gera as soluções iniciais aleatoriamente ou utilizando heurísticos conhecidos. Tais soluções são submetidas ao cruzamento e à seleção (para produzir novas soluções) e, somente após dez gerações, as soluções passam a ser melhoradas pelo *Simulated Annealing*, na sua segunda fase.

O método híbrido proposto no presente trabalho também gera as soluções iniciais usando métodos heurísticos conhecidos, além de utilizar o operador de cruzamento para gerar novas soluções. Porém, as duas melhores soluções são imediatamente submetidas ao *Simulated Annealing*.

No caso específico de problemas de programação de operações *Flow Shop*, temos o exemplo de MURATA, ISHIBUCHI E TANAKA (1996), que também propuseram um método heurístico híbrido Algoritmo Genético-*Simulated Annealing*. Eles utilizaram o Algoritmo Genético como base para o *Genetic Simulated Annealing (GSA)*. Porém, a diferença entre o *GSA* e o método proposto neste trabalho é que no *GSA*, o *Simulated Annealing* é inserido entre os passos inicialização (geração da população inicial) e seleção realizados pelo Algoritmo Genético, ao passo que no método híbrido proposto neste trabalho, o Algoritmo Genético e o *Simulated Annealing* operam em fases distintas.

4.3. O Algoritmo Genético Puro

(operadores de cruzamento *onpointGA* ou *pmxGA*)

PASSO 1) Seleção dos pais: obter a seqüência inicial 1 (solução do NEH) e a seqüência inicial 2 (solução inicial do FSHOPH).

$s_1 :=$ seqüência inicial 1

$s_2 :=$ seqüência inicial 2

$f(s_1) :=$ *makespan* da seqüência inicial 1

$f(s_2) :=$ *makespan* da seqüência inicial 2

Se $f(s_1) \leq f(s_2)$, então $BS := s_1$ {melhor solução já obtida}

e $BM := f(s_1)$ {*makespan* da melhor solução}

Caso contrário, $BS := s_2$ e $BM := f(s_2)$.

Enquanto o número total de seqüências geradas e avaliadas \leq TNbS(m,n)

e $s_1 \neq s_2$:

PASSO 2) Aplicar o operador de cruzamento em s_1 e s_2 :

$s_1 \times s_2 \rightarrow s_3$ e s_4 (onde s_3 e s_4 são as seqüências descendentes).

Aplicar o **operador de mutação** em s_3 e s_4 , se for o caso.

Ordenar as 4 seqüências em ordem não-decrescente do *makespan* $f(s)$.

PASSO 3) Selecionar as 2 seqüências que apresentam os 2 menores *makespans*

$s_1 := \text{seq}[1]$ {seqüência com o primeiro melhor *makespan*}

$f(s_1) := f(\text{seq}[1])$ {*makespan* da primeira melhor seqüência}

$s_2 := \text{seq}[2]$ {seqüência com o segundo melhor *makespan*}

$f(s_2) := f(\text{seq}[2])$ {*makespan* da segunda melhor seqüência}

Se $f(s_1) < \text{BM}$ então

$\text{BS} := \text{seq}[1]$

$\text{BM} := f(\text{seq}[1])$.

PASSO 4) Tomar as seqüências s_1 e s_2 e aplicar o procedimento descrito a partir do

Passo 2.

• **Resultados:**

BS : a melhor seqüência das tarefas (**solução do problema**);

BM : valor do *makespan* ou duração total da programação das tarefas segundo BS.

4.4. O Método *Simulated Annealing* Puro

PASSO 1) Obtenção da solução inicial s : obter a seqüência 1 (solução do NEH) e a seqüência 2 (solução inicial do FSHOPH).

$s_1 :=$ seqüência 1

$s_2 :=$ seqüência 2

$f(s_1) := \text{makespan}$ da seqüência 1

$f(s_2) := \text{makespan}$ da seqüência 2

Se $f(s_1) \leq f(s_2)$, então $s := s_1$

Caso contrário, $s := s_2$

$BS := s$ {melhor solução já obtida}

$BM := f(s)$ {*makespan* da melhor solução}.

Enquanto o número total de seqüências geradas e avaliadas $\leq TNbS(m,n)$:

PASSO 2) Obter aleatoriamente a seqüência s' a partir da vizinhança de inserção da seqüência atual s .

PASSO 3) Se $f(s') \leq f(s)$ então $s := s'$

Caso contrário, tomar ao acaso um valor R do intervalo $[0,1]$

Se $R \leq p(k)$, então $s := s'$

Atualizar o valor do parâmetro temperatura.

PASSO 4) Se $f(s) < BM$, então $BS := s$ e $BM := f(s)$.

Voltar ao **Passo 2**.

• **Resultados:**

BS : a melhor seqüência das tarefas (**solução do problema**);

BM : valor do *makespan* ou duração total da programação das tarefas segundo BS.

4.5. O Método Híbrido *HBGASA*

No método híbrido *HBGASA*, os passos 1, 2 e 3 referem-se à sua “parte Algoritmo genético”, enquanto que os *passos 4 e 5* (em *itálico*) correspondem à “parte *Simulated Annealing*”.

PASSO 1) Seleção dos pais: obter a seqüência inicial 1 (solução do NEH) e a seqüência inicial 2 (solução inicial do FSHOPH).

$s_1 :=$ seqüência inicial 1

$s_2 :=$ seqüência inicial 2

$f(s_1) :=$ *makespan* da seqüência inicial 1

$f(s_2) :=$ *makespan* da seqüência inicial 2

Se $f(s_1) \leq f(s_2)$, então $BS := s_1$ {melhor solução já obtida}

e $BM := f(s_1)$ {*makespan* da melhor solução}

Caso contrário, $BS := s_2$ e $BM := f(s_2)$.

Enquanto o número total de seqüências geradas e avaliadas \leq TNbS(m,n)

e $s_1 \neq s_2$:

PASSO 2) Aplicar o operador de cruzamento em s_1 e s_2 :

$s_1 \otimes s_2 \rightarrow s_3$ e s_4 (onde s_3 e s_4 são as seqüências descendentes)

Ordenar as 4 seqüências em ordem não-decrescente do *makespan* $f(s)$.

PASSO 3) Selecionar as 2 seqüências que apresentam os 2 menores *makespans*

$s_1 :=$ seq[1] {seqüência com o primeiro melhor *makespan*}

$f(s_1) :=$ $f(\text{seq}[1])$ {*makespan* da primeira melhor seqüência}

$s_2 :=$ seq[2] {seqüência com o segundo melhor *makespan*}

$f(s_2) :=$ $f(\text{seq}[2])$ {*makespan* da segunda melhor seqüência}

Se $f(s_1) < BM$ então

$BS :=$ seq[1]

$BM :=$ $f(\text{seq}[1])$.

PASSO 4) Aplicar a técnica Simulated Annealing tendo s_1 como solução inicial, com o número de iterações $K = Nvp(S)$, determinando a melhor seqüência s^* (a de menor makespan).

$$s_1 := s^*$$

$$f(s_1) := f(s^*)$$

Se $f(s_1) < BM$ então

$$BS := s_1$$

$$BM := f(s_1).$$

PASSO 5) Aplicar a técnica Simulated Annealing tendo s_2 como solução inicial, com o número de iterações $K = Nvp(S)$, determinando a melhor seqüência s^{} (a de menor makespan).**

$$s_2 := s^{**}$$

$$f(s_2) := f(s^{**})$$

Se $f(s_2) < BM$ então

$$BS := s_2$$

$$BM := f(s_2).$$

PASSO 6) Tomar as seqüências s_1 e s_2 e aplicar o procedimento descrito a partir do Passo 2.

• **Resultados:**

BS : a melhor seqüência das tarefas (**solução do problema**);

BM : valor do *makespan* ou duração total da programação das tarefas segundo BS.

→ **Observação:** deve-se notar que no método híbrido *HBGASA* e nos Algoritmos Genéticos puros, a busca por melhores soluções pode ser encerrada antes da condição de parada dada pelo número total de seqüências $TNbS(m,n)$. Isto ocorre quando as seqüências sobre as quais for aplicado o operador de cruzamento forem iguais (condição de parada secundária).

CAPÍTULO 5

EXPERIMENTAÇÃO COMPUTACIONAL E ANÁLISE DOS RESULTADOS OBTIDOS

5.1. Experimentação Computacional

A experimentação computacional foi dividida em duas etapas:

① Na primeira etapa, testou-se o método híbrido Algoritmo Genético-*Simulated Annealing* (HBGASA), utilizando-se o operador de cruzamento de UM CORTE. Foram desenvolvidos 100 programas híbridos distintos, através da combinação das diferentes temperaturas iniciais (T_1) e parcelas da vizinhança analisada (em função do parâmetro p). Tais programas foram comparados com o método metaheurístico Algoritmo Genético puro (denominado onecutGA) - que obviamente também utilizou o operador de cruzamento de UM CORTE - e com o método metaheurístico *Simulated Annealing* puro (denominado ModSAfshopH).

② Na segunda etapa, testou-se o método *HBGASA*, utilizando-se o operador de cruzamento *PMX* (*Partially Matched Crossover*). Também nesta etapa, foram desenvolvidos 100 programas híbridos, combinando-se as diferentes temperaturas iniciais e parcelas da vizinhança analisada. Estes 100 programas foram comparados com os métodos Algoritmo Genético puro (*pmxGA*) - que obviamente também utilizou o operador de cruzamento *PMX* - e com o *Simulated Annealing* puro (*ModSAfshopH*).

Em cada etapa foram testados 540 problemas, divididos em 27 classes de acordo com o número de máquinas (*m*) e o número de tarefas (*n*), para: ***m* = 4, 7 e 10 máquinas** e ***n* = 20, 30, 40, 50, 60, 70, 80, 90 e 100 tarefas**. Foram testados 20 problemas em cada classe (*m,n*).

Nas duas etapas, as temperaturas iniciais (*T₁*) utilizadas nos algoritmos híbridos foram: ***T₁* = {20; 30; 45; 60 e 70}**. Os valores testados do parâmetro *p* foram: ***p* = {0.05; 0.10; 0.15; 0.20; 0.25; 0.30; 0.35; 0.40; 0.45; 0.50; 0.55; 0.60; 0.65; 0.70; 0.75; 0.80; 0.85; 0.90; 0.95 e 1.00}**. Combinando-se as diferentes *T₁* e *p*, tem-se os 100 métodos *HBGASA* que empregam o operador de cruzamento de UM CORTE e os outros 100 métodos *HBGASA* que utilizam o operador de cruzamento *PMX*.

Todos os problemas foram gerados aleatoriamente, sendo os tempos de processamento das tarefas números inteiros distribuídos uniformemente no intervalo [1, 100]. O equipamento utilizado foi um microcomputador Pentium II - Intel MMX de 266 MHz, pertencente ao Laboratório de Multimídia e Processamento Científico da Área de Engenharia de Produção, Escola de Engenharia de São Carlos - Universidade de São Paulo.

Para que a experimentação fosse conduzida de maneira mais adequada e eficiente, desenvolveu-se um *software* (em linguagem Pascal) chamado “Gerenciador de Problemas *Flow Shop*”, contendo os algoritmos *onecutGA/pmxA*, *ModSAfshopH* e *HBGASA* (operador de cruzamento UM CORTE/*PMX*).

O menu principal do *software* é constituído pelas seguintes opções:

- ☞ GERAR aleatoriamente um ou mais problemas;
- ☞ Mostrar/Imprimir DADOS de um problema;
- ☞ Mostrar/Imprimir RESULTADOS de um problema;
- ☞ Calcular ESTATÍSTICAS de uma classe;
- ☞ Calcular ESTATÍSTICAS em função do número de tarefas;
- ☞ RESOLVER um ou mais problemas e
- ☞ SAIR do *HBGASA1XGAmoSA.Flowshop*
(ou SAIR do *HBGASApmxGAmoSA.Flowshop*).

Os resultados obtidos na experimentação e os cálculos estatísticos realizados pelo *software* foram, portanto, divididos em duas categorias distintas:

- i) máquinas não-agrupadas (opção “Calcular ESTATÍSTICAS de uma classe”) e
- ii) máquinas agrupadas (opção “Calcular ESTATÍSTICAS em função do número de tarefas”).

Os resultados da Experimentação Computacional foram obtidos utilizando-se as seguintes estatística : *Porcentagem de Sucesso, Melhoria Relativa, Índice de Desempenho da Busca e Tempo de Computação*.

A *Porcentagem de Sucesso* é calculada pelo número de vezes em que o método obteve a melhor solução, dividido pelo número total de problemas avaliados. Obviamente, quando dois ou mais métodos obtêm a melhor solução para um mesmo problema todos eles alcançam sucesso e conseqüentemente suas porcentagens de sucesso são simultaneamente melhoradas.

A *Melhoria Relativa* mede o percentual de redução da Duração Total da Programação, em relação à solução inicial. Nos métodos que utilizam o Algoritmo Genético, a solução inicial considerada para o cálculo da melhoria relativa é aquela que fornece o menor *makespan* dentre a solução inicial do FSHOPH e a solução do

NEH. É importante ressaltar que todos os métodos comparados entre si têm a mesma solução inicial. A Melhoria Relativa é obtida pela seguinte expressão :

$$RI = (M1 - M) / M1 \quad \text{onde:}$$

- M1: *makespan* da solução inicial e
- M : *makespan* da melhor seqüência que foi encontrada pelo método heurístico.

O *Índice de Desempenho da Busca* é o quociente entre o número de vezes em que o método conseguiu melhorar a *solução corrente* do processo de busca e o número de seqüências a serem avaliadas de acordo com a Condição de Parada.

Hierarquicamente, a estatística que deverá orientar a escolha do melhor método é a *Porcentagem de Sucesso*. As outras duas são a ela correlacionadas e deverão ser usadas para reiterar o desempenho quanto à estatística principal. Em outras palavras, quando um método apresenta valores superiores da porcentagem de sucesso, é *estatisticamente esperado* que a melhoria relativa e o índice de desempenho da busca também apresentem valores superiores.

Quanto ao *Tempo de Computação*, tendo em vista que os métodos foram projetados para não apresentarem tempos excessivos e grandes diferenças para problemas da mesma classe (m, n), eles são calculados apenas com o propósito de avaliação dos esforços computacionais.

5.2. Análise dos Resultados Obtidos

Os resultados obtidos nas duas etapas da experimentação computacional foram analisados de acordo com o seguinte procedimento:

i) Inicialmente, tomou-se os resultados de Porcentagens de Sucesso para máquinas agrupadas ($m = 4, 7$ e 10 máquinas e $n = 20$ a 100 tarefas). As Porcentagens de Sucesso obtidas por cada um dos 100 algoritmos *HBGASA* (operador de cruzamento UM CORTE/PMX) foram comparadas às Porcentagens de Sucesso obtidas pelos métodos Algoritmo Genético puro (operador de cruzamento UM CORTE/PMX) e *Simulated Annealing* puro;

ii) Em seguida, o método híbrido *HBGASA* de melhor desempenho (quando comparado ao Algoritmo Genético e *Simulated Annealing* puros) obtido na fase (i), foi analisado em termos de resultados estatísticos para máquinas não-agrupadas.

5.2.1. Método Híbrido *HBGASA* com operador de cruzamento de UM CORTE

As Tabelas 2 a 6 ilustram os resultados obtidos na comparação das Porcentagens de Sucesso para máquinas agrupadas ($m = 4, 7$ e 10), em relação aos diferentes algoritmos nas temperaturas iniciais consideradas ($T_1 = 20, 30, 45, 60$ e 70).

Na elaboração das tabelas citadas acima, foi utilizada a seguinte estrutura:

⊖ o elemento “X” indica que o método híbrido *HBGASA* obteve maior Porcentagem de Sucesso para uma determinada parcela da vizinhança analisada, quando comparado aos métodos Algoritmo Genético e *Simulated Annealing* puros;

⊖ o elemento “•” indica que o método *HBGASA* apresentou a mesma Porcentagem de Sucesso obtida pelo *Simulated Annealing* puro (empate);

⊖ o elemento “*” indica que o Algoritmo Genético puro obteve maior Porcentagem de Sucesso, quando comparado aos demais métodos e

⊖ célula não preenchida indica que o *Simulated Annealing* puro obteve maior Porcentagem de Sucesso, quando comparado aos demais métodos.

No caso dos métodos híbridos, os dois primeiros números encontrados junto ao nome “*HBGASA*” indicam a temperatura inicial (T_1) e os dois números seguintes indicam o valor do parâmetro p .

Exemplo:

⇒ *HBGASA* 2010 ⇒ $T_1 = 20$ e $p = 10\%$.

TABELA 2: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e *HBGASA* ($T_1 = 20$)

Algoritmo	Número de Tarefas									Vitórias <i>HBGASA</i>
	20	30	40	50	60	70	80	90	100	
<i>HBGASA</i> 2005							X		X	2
<i>HBGASA</i> 2010						•	X			1
<i>HBGASA</i> 2015							X	X	X	3
<i>HBGASA</i> 2020				X	X	X	X	X	X	6
<i>HBGASA</i> 2025		X		X	•	X	X	X	X	6
<i>HBGASA</i> 2030			X		X	X	X	X	X	6
<i>HBGASA</i> 2035		X		X	•	X	X	X	X	6
<i>HBGASA</i> 2040			X		X	X	X	X	X	6
<i>HBGASA</i> 2045					X		X	X	X	4
<i>HBGASA</i> 2050		X	X	•	X		X	X	X	6
<i>HBGASA</i> 2055		X	X	X	X		X	X	X	7
<i>HBGASA</i> 2060		X	X	•	X	X	X		X	6
<i>HBGASA</i> 2065		X	•		•	X	X	X	X	5
<i>HBGASA</i> 2070	X	X		X	X	X	X	X	X	8
<i>HBGASA</i> 2075	X		X	X	X	X	X	X	X	8
<i>HBGASA</i> 2080					X		X	X	X	4
<i>HBGASA</i> 2085	X	X	X		X	X	X	X	X	8
<i>HBGASA</i> 2090	•	X		X	X		X	X	X	6
<i>HBGASA</i> 2095	X			X	X	X	X	X	X	7
<i>HBGASA</i> 20100			X	X	X		X	X	X	6
Vitórias <i>HBGASA</i>	4	9	8	9	14	11	20	17	19	

TABELA 3: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e HBGASA ($T_1 = 30$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 3005									X	1
HBGASA 3010									X	1
HBGASA 3015						X	X		X	3
HBGASA 3020			•						X	1
HBGASA 3025			X						X	2
HBGASA 3030				X			X		X	3
HBGASA 3035						•			X	1
HBGASA 3040						•	X			1
HBGASA 3045									X	1
HBGASA 3050					•				X	1
HBGASA 3055	•									0
HBGASA 3060							X		X	2
HBGASA 3065		•						X	X	2
HBGASA 3070							X	X	X	3
HBGASA 3075										0
HBGASA 3080							X			1
HBGASA 3085									•	0
HBGASA 3090			X					X		2
HBGASA 3095							X			1
HBGASA 30100							•	X	X	2
Vitórias HBGASA	0	0	2	1	0	1	7	4	13	

TABELA 4: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e HBGASA ($T_1 = 45$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 4505										0
HBGASA 4510										0
HBGASA 4515								X	X	2
HBGASA 4520							X		X	2
HBGASA 4525										0
HBGASA 4530							•			0
HBGASA 4535									X	1
HBGASA 4540									X	1
HBGASA 4545							X		X	2
HBGASA 4550							X		X	2
HBGASA 4555								•		0
HBGASA 4560								X		1
HBGASA 4565										0
HBGASA 4570									•	0
HBGASA 4575								X	X	2
HBGASA 4580									•	0
HBGASA 4585								•		0
HBGASA 4590							•			0
HBGASA 4595										0
HBGASA 45100							•	X	•	1
Vitórias HBGASA	0	0	0	0	0	0	3	4	7	

TABELA 5: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e *HBGASA* ($T_1 = 60$)

Algoritmo	Número de Tarefas									Vitórias <i>HBGASA</i>
	20	30	40	50	60	70	80	90	100	
<i>HBGASA</i> 6005							X			1
<i>HBGASA</i> 6010										0
<i>HBGASA</i> 6015									X	1
<i>HBGASA</i> 6020										0
<i>HBGASA</i> 6025							X			1
<i>HBGASA</i> 6030										0
<i>HBGASA</i> 6035										0
<i>HBGASA</i> 6040										0
<i>HBGASA</i> 6045									X	1
<i>HBGASA</i> 6050										0
<i>HBGASA</i> 6055										0
<i>HBGASA</i> 6060									X	1
<i>HBGASA</i> 6065										0
<i>HBGASA</i> 6070										0
<i>HBGASA</i> 6075										0
<i>HBGASA</i> 6080										0
<i>HBGASA</i> 6085										0
<i>HBGASA</i> 6090									X	1
<i>HBGASA</i> 6095										0
<i>HBGASA</i> 60100										0
Vitórias <i>HBGASA</i>	0	0	0	0	0	0	2	0	4	

TABELA 6: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH e *HBGASA* ($T_1 = 70$)

Algoritmo	Número de Tarefas									Vitórias <i>HBGASA</i>
	20	30	40	50	60	70	80	90	100	
<i>HBGASA</i> 7005										0
<i>HBGASA</i> 7010										0
<i>HBGASA</i> 7015									X	1
<i>HBGASA</i> 7020										0
<i>HBGASA</i> 7025									X	1
<i>HBGASA</i> 7030							X			1
<i>HBGASA</i> 7035										0
<i>HBGASA</i> 7040										0
<i>HBGASA</i> 7045							X			1
<i>HBGASA</i> 7050										0
<i>HBGASA</i> 7055										0
<i>HBGASA</i> 7060										0
<i>HBGASA</i> 7065										0
<i>HBGASA</i> 7070										0
<i>HBGASA</i> 7075										0
<i>HBGASA</i> 7080										0
<i>HBGASA</i> 7085										0
<i>HBGASA</i> 7090										0
<i>HBGASA</i> 7095										0
<i>HBGASA</i> 70100										0
Vitórias <i>HBGASA</i>	0	0	0	0	0	0	2	0	0	

A comparação das Porcentagens de Sucesso para máquinas agrupadas obtidas na experimentação computacional em relação aos métodos *HBGASA* (operador de cruzamento de UM CORTE), *onecutGA* e *ModSAfshopH* (Tabelas 2 a 6), evidencia alguns aspectos importantes:

① O Algoritmo Genético puro (*onecutGA*) foi superado pelo *Simulated Annealing* puro (*ModSAfshopH*) e/ou pelo método híbrido *HBGASA* em todas as classes de problemas avaliadas, ou seja, em nenhuma situação envolvendo os diferentes valores de T_1 e p , o método *onecutGA* apresentou Porcentagem de Sucesso superior ao *ModSAfshopH* ou a qualquer método *HBGASA*.

Isso é explicado pela seguinte razão: o *onecutGA* cessa sua busca pela melhor solução (de menor *makespan*) a partir do momento em que as “seqüências-pais” geradas passam a ser idênticas. Isso pode acontecer após algumas iterações. Portanto, em determinadas situações, o critério de parada pode ser atingido rapidamente, uma vez que o método já parte de duas soluções iniciais de boa qualidade. Com isso, um número relativamente pequeno de seqüências é efetivamente avaliado, o que diminui a possibilidade de se encontrar soluções de melhor qualidade.

② O método híbrido *HBGASA* superou o método *ModSAfshopH* somente quando a temperatura inicial foi baixa (mais especificamente, quando $T_1 = 20$).

Nas demais temperaturas, o método *ModSAfshopH* apresentou desempenho bastante superior, quando comparado ao *HBGASA* e ao *onecutGA*.

③ O desempenho do *HBGASA* melhorou em função do aumento do número de tarefas do problema.

O número de seqüências analisadas pelos métodos aumenta de acordo com o número de tarefas e máquinas do problema. Portanto, quanto maior o tamanho do problema, maior é o número de seqüências vizinhas analisadas e, portanto, maiores são as possibilidades de “sucesso” do método.

De acordo com os resultados encontrados nas Tabelas 2 a 6, pode-se observar que os métodos híbridos *HBGASA* que apresentaram os melhores desempenhos

(quando comparados aos métodos onecutGA e ModSAfshopH), foram aqueles que trabalharam com $T_1 = 20$ (vide Tabela 2). Dentre eles, destacam-se:

➔ **HBGASA 2070** ($T_1 = 20$ e $p = 70\%$);

➔ **HBGASA 2075** ($T_1 = 20$ e $p = 75\%$) e

➔ **HBGASA 2085** ($T_1 = 20$ e $p = 85\%$).

A comparação entre as Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModSAfshopH, *HBGASA 2070*, *HBGASA 2075* e *HBGASA 2085* (operador de cruzamento de UM CORTE) para máquinas agrupadas é encontrada nas Tabelas 7 a 9. Os mesmos resultados estão representados graficamente nas Figuras 4 a 6.

TABELA 7: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModSAfshopH X HBGASA 2070)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
onecutGA	21,66	21,66	28,33	21,66	26,66	34,99	33,33	33,33	43,33
ModSAfshopH	68,33	68,33	80,00	71,66	63,33	64,99	66,66	71,66	60,00
HBGASA 2070	78,33	73,33	71,66	75,00	80,00	81,66	83,33	81,66	89,99

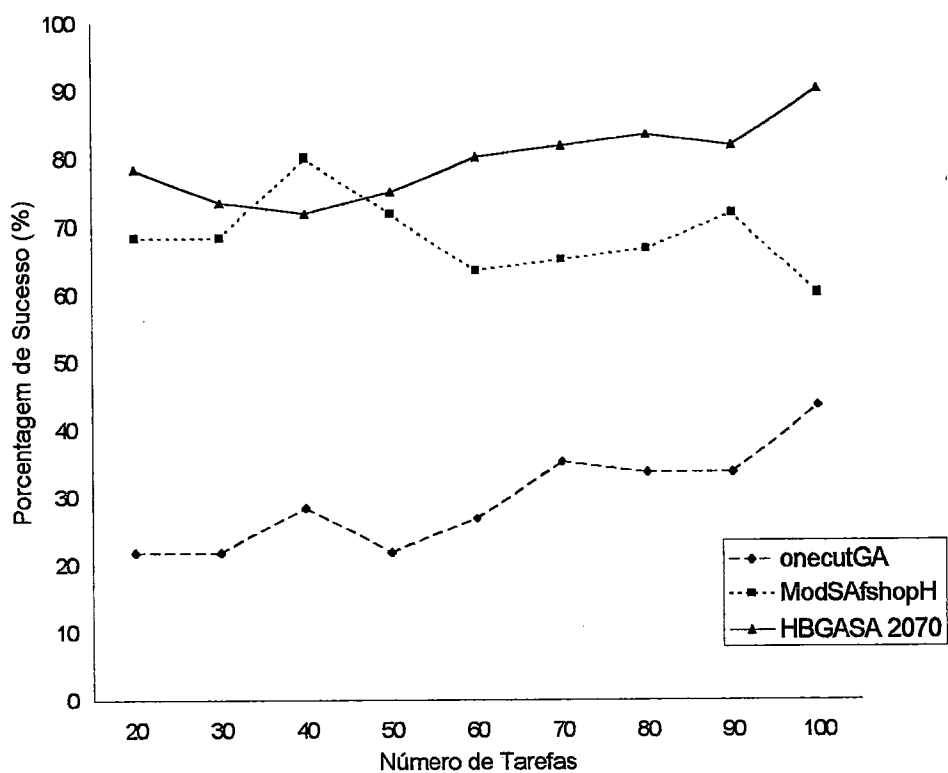


FIGURA 4: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModSAfshopH X HBGASA 2070)

TABELA 8: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModSAfshopH X HBGASA 2075)

Algoritmo	Número de Tarefas									
	20	30	40	50	60	70	80	90	100	
onecutGA	21,66	21,66	25,00	21,66	28,33	34,99	31,66	31,66	46,66	
ModSAfshopH	73,33	69,99	66,66	71,66	66,66	71,66	71,66	73,33	71,66	
HBGASA 2075	78,33	64,99	76,66	78,33	78,33	76,66	76,66	83,33	81,66	

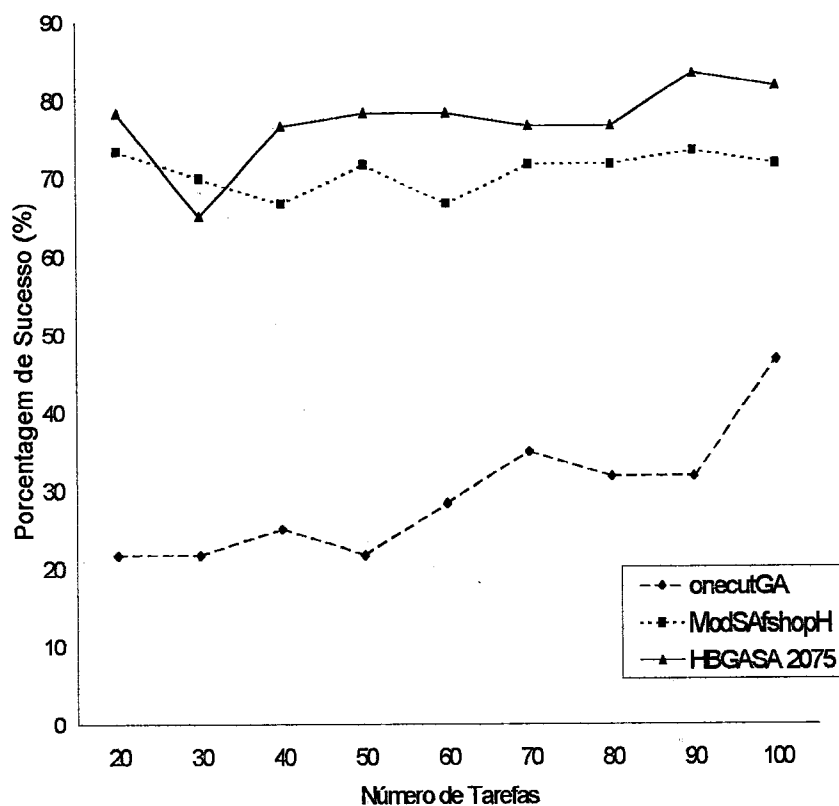


FIGURA 5: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModSAfshopH X HBGASA 2075)

TABELA 9: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModSAfshopH X HBGASA 2085)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
onecutGA	20,00	20,00	23,33	21,66	31,66	38,33	31,66	34,99	44,99
ModSAfshopH	69,99	66,66	68,33	81,66	73,33	71,66	73,33	75,00	68,33
HBGASA 2085	76,66	68,33	73,33	63,33	76,66	78,33	75,00	85,00	81,66

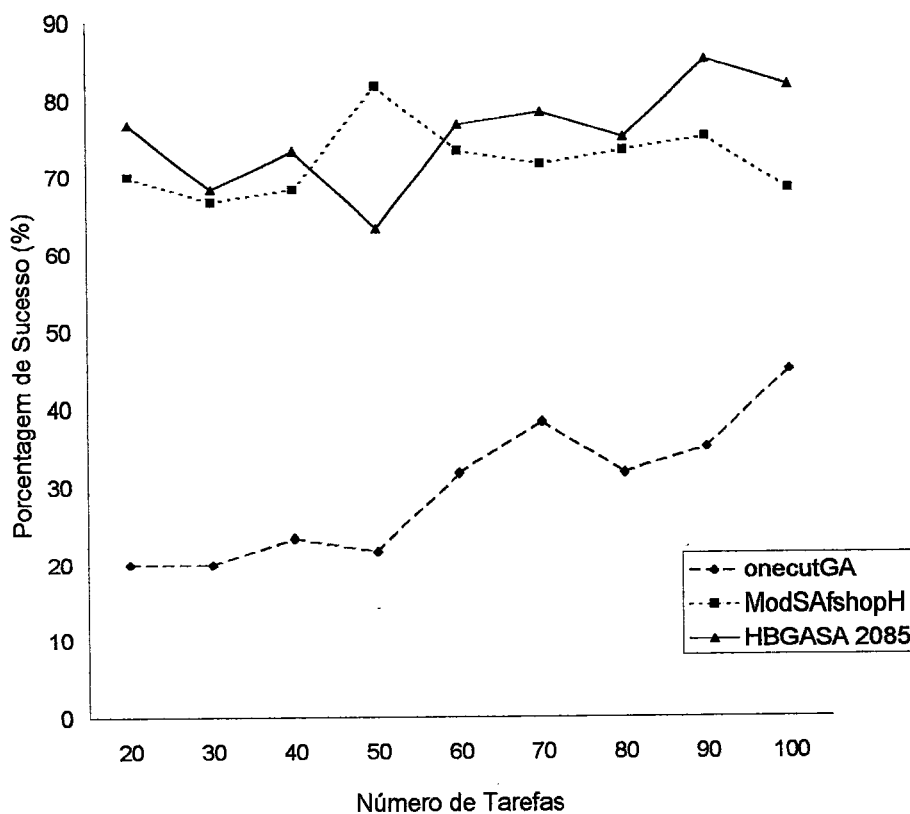


FIGURA 6: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModSAfshopH X HBGASA 2085)

Os métodos híbridos *HBGASA 2070*, *HBGASA 2075* e *HBGASA 2085* se destacaram dentre os 100 métodos híbridos que empregaram o operador de cruzamento de UM CORTE. Apesar do bom desempenho por eles atingido, as Tabelas 7 a 9 e as Figuras 4 a 6 também mostram que:

⇒ o método *HBGASA 2070* foi superado pelo ModSAfshopH no caso dos problemas com $n = 40$ tarefas (vide Figura 4);

⇒ o método *HBGASA 2075* foi superado pelo ModSAfshopH no caso dos problemas com $n = 30$ tarefas (vide Figura 5) e

⇒ o método *HBGASA 2085* foi superado pelo ModSAfshopH no caso dos problemas com $n = 50$ tarefas (vide Figura 6).

5.2.2. Método Híbrido *HBGASA* com operador de cruzamento PMX

O método *HBGASA* com operador de cruzamento PMX foi comparado com os métodos pmxGA (Algoritmo Genético com operador de cruzamento PMX) e ModSAfshopH.

As Tabelas 10 a 14 ilustram a comparação das Porcentagens de Sucesso para máquinas agrupadas, obtidas pelos métodos pmxGA, ModSAfshopH e *HBGASA* (com operador de cruzamento PMX), em relação às temperaturas iniciais $T_1 = 20, 30, 45, 60$ e 70 . A estrutura empregada nestas tabelas é idêntica à estrutura utilizada nas Tabelas 2 a 6.

TABELA 10: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e HBGASA ($T_1 = 20$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 2005			X	X	X	X	X	X	X	7
HBGASA 2010		X	X	X	X	X	X	X	X	8
HBGASA 2015		X	X	X	X	X	X	X	X	8
HBGASA 2020	X	X	X	X	X	X	X	X	X	9
HBGASA 2025	X	X	X	X	X	X	X	X	X	9
HBGASA 2030	X	X	X	X	X	X	X	X	X	9
HBGASA 2035	X	X	X	X	X	X	X	X	X	9
HBGASA 2040	X	X	X	X	X	X	X	X	X	9
HBGASA 2045	X	X	X	X	X	X	X	X	X	9
HBGASA 2050	X	X	X	X	X	X	X	X	X	9
HBGASA 2055	X	X	X	X	X	X	X	X	X	9
HBGASA 2060	X	X	X	X	X	X	X	X	X	9
HBGASA 2065	X	X	X	X	X	X	X	X	X	9
HBGASA 2070	X	X	X	X	X	X	X	X	X	9
HBGASA 2075	X	X	X	X	X	X	X	X	X	9
HBGASA 2080	X	X	X	X	X	X	X	X	X	9
HBGASA 2085	X	X	X	X	X	X	X	X	X	9
HBGASA 2090	X	X	X	X	X	X	X	X	X	9
HBGASA 2095	X	X	X	X	X	X	X	X	X	9
HBGASA 20100	X	X	X	X	X	X	X	X	X	9
Vitórias HBGASA	17	19	20	20	20	20	20	20	20	

TABELA 11: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e HBGASA ($T_1 = 30$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 3005		X	X	X	X	X	X	X	X	8
HBGASA 3010		X	X	X	X	X	X	X	X	8
HBGASA 3015	X	X	X	X	X	X	X	X	X	9
HBGASA 3020		X	X	X	X	X	X	X	X	8
HBGASA 3025	X	X	X	X	X	X	X	X	X	9
HBGASA 3030	X	X	X	X	X	X	X	X	X	9
HBGASA 3035	X	X	X	X	X	X	X	X	X	9
HBGASA 3040	X	X	X	X	X	X	X	X	X	9
HBGASA 3045	X	X	X	X	X	X	X	X	X	9
HBGASA 3050	X	X	X	X	X	X	X	X	X	9
HBGASA 3055	X	X	X	X	X	X	X	X	X	9
HBGASA 3060	X	X	X	X	X	X	X	X	X	9
HBGASA 3065	X	X	X	X	X	X	X	X	X	9
HBGASA 3070	X	•	X	X	X	X	X	X	X	8
HBGASA 3075	X	X	X	X	X	X	X	X	X	9
HBGASA 3080	X	X	X	X		X	X	X	X	8
HBGASA 3085	X	X	X	X	X	X	X	X	X	9
HBGASA 3090	X		X	X	X	X	X	X	X	8
HBGASA 3095	X	•	X	X	X	X	X	X	X	8
HBGASA 30100	X		X	X	X	X	X	X	X	8
Vitórias HBGASA	17	16	20	20	19	20	20	20	20	

TABELA 12: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e HBGASA ($T_1 = 45$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 4505				X	X	X	X		X	5
HBGASA 4510				X	X	X	X	X	X	6
HBGASA 4515		X	X	X	X	X	X	X	X	8
HBGASA 4520	X	X	X	X	X	X	X	X	X	9
HBGASA 4525	X	X	X	X	X	X	X	X	X	9
HBGASA 4530	X	X	X	X	X	X	X	X	X	9
HBGASA 4535	X		X	X	X	X	X	X	X	8
HBGASA 4540	X	X	X	X	X	X	X	X	X	9
HBGASA 4545	X		X		X	X	X	X	X	7
HBGASA 4550	X	X		X	X	X	X	X	X	8
HBGASA 4555	X	X	X	X	X	X	X	X	X	9
HBGASA 4560	X		•	X	X	X	X	X	X	7
HBGASA 4565	X	X		X	X	X	X	X	X	8
HBGASA 4570				X	X	X	X	X	X	6
HBGASA 4575				•	X		X	X	X	4
HBGASA 4580	•	X		X		X		X	•	4
HBGASA 4585	X	X		X		X		X	X	6
HBGASA 4590	•		X	X		X	X	X		5
HBGASA 4595			•	X		X				2
HBGASA 45100			•		X	X				2
Vitórias HBGASA	11	10	9	17	16	19	16	17	16	

TABELA 13: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e HBGASA ($T_1 = 60$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 6005				X	X	X	X	X	X	6
HBGASA 6010				X		X	X	X	X	5
HBGASA 6015		X	X	X	X	X	X	X	X	8
HBGASA 6020		X		X		X	X	X	X	6
HBGASA 6025		X		X	X	X	X	X	X	7
HBGASA 6030	X	X	X	X	X	X	X	X	X	9
HBGASA 6035		X	X	•	X	X	X	X	X	7
HBGASA 6040	X	X		X	X	X		X	X	7
HBGASA 6045	X	X		X	X		X	X	X	7
HBGASA 6050	X	X		X		X		X	X	5
HBGASA 6055	X		X			X	X	X	X	6
HBGASA 6060	•			X	X			X	X	4
HBGASA 6065	•							•		0
HBGASA 6070	X		X		X				X	4
HBGASA 6075	X		X			X		•	•	3
HBGASA 6080	X					X	X		X	4
HBGASA 6085		•					X			1
HBGASA 6090	X								•	1
HBGASA 6095						X	X	•	X	3
HBGASA 60100	X							X		2
Vitórias HBGASA	10	8	6	10	9	13	12	12	15	

TABELA 14: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModSAfshopH e HBGASA ($T_1 = 70$)

Algoritmo	Número de Tarefas									Vitórias HBGASA
	20	30	40	50	60	70	80	90	100	
HBGASA 7005				X	X	X	X	X	X	6
HBGASA 7010				X	X	X	X	X	X	6
HBGASA 7015		X		X	X	X	X	X	X	7
HBGASA 7020		X	X	X	X	X	X	X	X	8
HBGASA 7025		X	X	X	X	X	X		X	7
HBGASA 7030		X	X	X	X	X	X	X	X	8
HBGASA 7035	X	X				X	X	X	X	6
HBGASA 7040		•		X	•	X		X	X	4
HBGASA 7045	X	X		X		X	X	X	X	7
HBGASA 7050	X					•		X	X	3
HBGASA 7055							X	X	X	3
HBGASA 7060	•						X	X	X	3
HBGASA 7065							X		X	2
HBGASA 7070							X		X	2
HBGASA 7075						X	X		X	3
HBGASA 7080									X	1
HBGASA 7085									X	1
HBGASA 7090					X			X	X	3
HBGASA 7095									•	0
HBGASA 70100							X		•	1
Vitórias HBGASA	3	6	3	8	7	10	14	12	18	

A comparação entre as Porcentagens de Sucesso para máquinas agrupadas obtidas pelo método híbrido HBGASA com operador de cruzamento PMX e pelos métodos puros (Tabelas 10 a 14), apresenta alguns aspectos importantes:

① O método HBGASA com operador de cruzamento PMX apresentou desempenho superior em relação ao desempenho do HBGASA com operador de UM CORTE (vide Tabelas 2 a 6), para todas as faixas de temperatura inicial (T_1) e parcela da vizinhança analisada (p).

Devido às características particulares de funcionamento (vide Capítulo 4), o operador de cruzamento PMX é capaz de gerar uma maior variedade de vizinhos (quando comparado com o operador de cruzamento de UM CORTE), tornando-o mais eficaz e apropriado para uso no problema tratado neste trabalho.

② O método híbrido HBGASA com operador de cruzamento PMX apresentou desempenho superior quando comparado ao ModSAfshopH, nos

seguintes casos: $T_1 = 20$, $T_1 = 30$, $T_1 = 45$ e $T_1 = 60$. O ModSAfshopH superou o híbrido *HBGASA* somente quando $T_1 = 70$ ($p = 0,95$);

③ O Algoritmo Genético puro (pmxGA) foi superado pelo *Simulated Annealing* puro (ModSAfshopH) e/ou pelo método híbrido *HBGASA* em todas as classes de problemas (m,n), nas diferentes temperaturas iniciais. Isto é, o método pmxGA foi superado pelos métodos ModSAfshopH e/ou *HBGASA* em todas as situações analisadas. Este comportamento já havia sido observado em relação ao Algoritmo Genético com operador de cruzamento de UM CORTE;

④ De maneira semelhante ao comportamento apresentado pelo método *HBGASA* com operador de cruzamento de UM CORTE, o *HBGASA* com operador PMX apresentou melhor desempenho quando $T_1 = 20$ e $T_1 = 30$;

⑤ O desempenho do método *HBGASA* melhorou em função do aumento do número de tarefas do problema, ou seja, quanto maior o tamanho do problema, maior a Porcentagem de Sucesso obtida pelo *HBGASA*, quando comparada ao ModSAfshopH e pmxGA;

Vale ressaltar dois aspectos fundamentais:

☞ O método híbrido com operador de cruzamento PMX apresentou desempenho bastante superior, quando comparado ao desempenho do método *HBGASA* com operador de cruzamento de UM CORTE ✓

☞ O método *HBGASA* com operador PMX obteve melhor desempenho quando trabalhou com baixas temperaturas iniciais ✓

Dentro deste contexto, os métodos híbridos que se destacaram foram os seguintes: ***HBGASA 2030*** ($T_1 = 20$ e $p = 30\%$); ***HBGASA 3030*** ($T_1 = 30$ e $p = 30\%$); ***HBGASA 4530*** ($T_1 = 45$ e $p = 30\%$); ***HBGASA 6030*** ($T_1 = 60$ e $p = 30\%$) e ***HBGASA 7030*** ($T_1 = 70$ e $p = 30\%$).

Na segunda fase da análise dos resultados obtidos, estes cinco métodos híbridos foram analisados e comparados entre si, para a obtenção do melhor método *HBGASA*. Para isso, foram analisados os resultados obtidos pelos métodos anteriormente relacionados, para máquinas não-agrupadas.

A Tabela 15 mostra a comparação entre as Porcentagens de Sucesso da classe, para máquinas não-agrupadas, obtidas pelos métodos *HBGASA* 2030, *HBGASA* 3030, *HBGASA* 4530, *HBGASA* 6030 e *HBGASA* 7030. A estrutura empregada na sua construção é semelhante à estrutura utilizada nas Tabelas 2 a 6.

TABELA 15: Comparação das Porcentagens de Sucesso da classe ($p = 30\%$)

Algoritmo	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
<i>HBGASA</i> 2030	4	X	•	X	X	X	X	X	X	X
	7	X	X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X	X
<i>HBGASA</i> 3030	4	X	•		X	X	X	X		X
	7	X	X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X	X
<i>HBGASA</i> 4530	4	X		X	X	X	X	X	•	X
	7	X	X	X	X	X	X	X	X	X
	10		X	X	X		X	•	X	X
<i>HBGASA</i> 6030	4	X		X	X	X	X	•	X	X
	7	X	X	X	•	X	X	X	X	X
	10	X		X	X	X	X	X	X	•
<i>HBGASA</i> 7030	4			X	X	X	X	X	X	X
	7	•	X	X	X	X	•	X	X	X
	10		•	X	X	X	X	X	X	X

Como afirmado anteriormente, cada método foi empregado na resolução de 20 problemas para cada classe, em um total de 27 classes ($m = 4, 7$ e 10 X $n = 20$ a 100). Analisando-se os métodos acima selecionados, agora em função do número de vitórias/empates/derrotas sobre o método ModSAfshopH, têm-se os resultados apresentados na Tabela 16.

A Tabela 16 apresenta a seguinte estrutura:

⊖ Número de Vitórias: indica o número de vezes em que o método *HBGASA* apresentou Porcentagem de Sucesso superior quando comparado ao *ModSAfshopH*, em relação às 27 classes de problemas;

⊖ Número de Empates: indica o número de vezes em que os métodos *HBGASA* e *ModSAfshopH* apresentaram a mesma Porcentagem de Sucesso e

⊖ Número de Derrotas: indica o número de vezes em que o método *HBGASA* apresentou valores de Porcentagem de Sucesso inferiores aos valores apresentados pelo método *ModSAfshopH*.

TABELA 16: Número de vitórias, empates e derrotas do método *HBGASA* ($p = 30\%$)

Método Híbrido	Número de Vitórias	Número de Empates	Número de Derrotas	Total
<i>HBGASA</i> 2030	26	1	0	27
<i>HBGASA</i> 3030	24	1	2	27
<i>HBGASA</i> 4530	22	2	3	27
<i>HBGASA</i> 6030	22	3	2	27
<i>HBGASA</i> 7030	21	3	3	27

As Tabelas 15 e 16 mostram que o método *HBGASA* 2030 se destacou dentre todos os métodos híbridos, uma vez que obteve maior número de vitórias - em relação às Porcentagens de Sucesso - sobre o *ModSAfshopH* (26 vitórias e apenas 1 empate).

O ótimo desempenho frente aos métodos *pmxGA*, *ModSAfshopH* e aos demais métodos híbridos, indica que o método *HBGASA* 2030, com operador de cruzamento *PMX*, é o melhor dentre todos os métodos puros e/ou híbridos desenvolvidos e testados no presente trabalho.

As Porcentagens de Sucesso obtidas pelos métodos *pmxGA*, *ModSAfshopH* e *HBGASA* 2030 para máquinas não-agrupadas e máquinas agrupadas ($T_1 = 20$) estão representadas graficamente nas Figuras 7 a 9 e Figura 10, respectivamente.

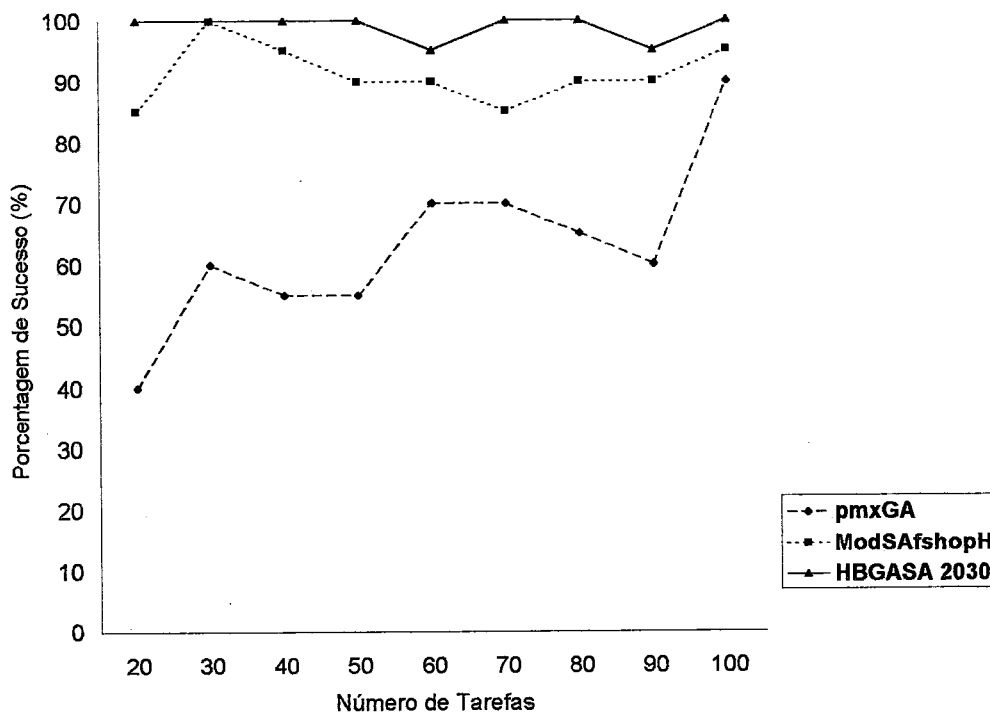


FIGURA 7: Percentagens de Sucesso para máquinas não-agrupadas (m = 4)

pmxGA X ModSAfshopH X HBGASA 2030

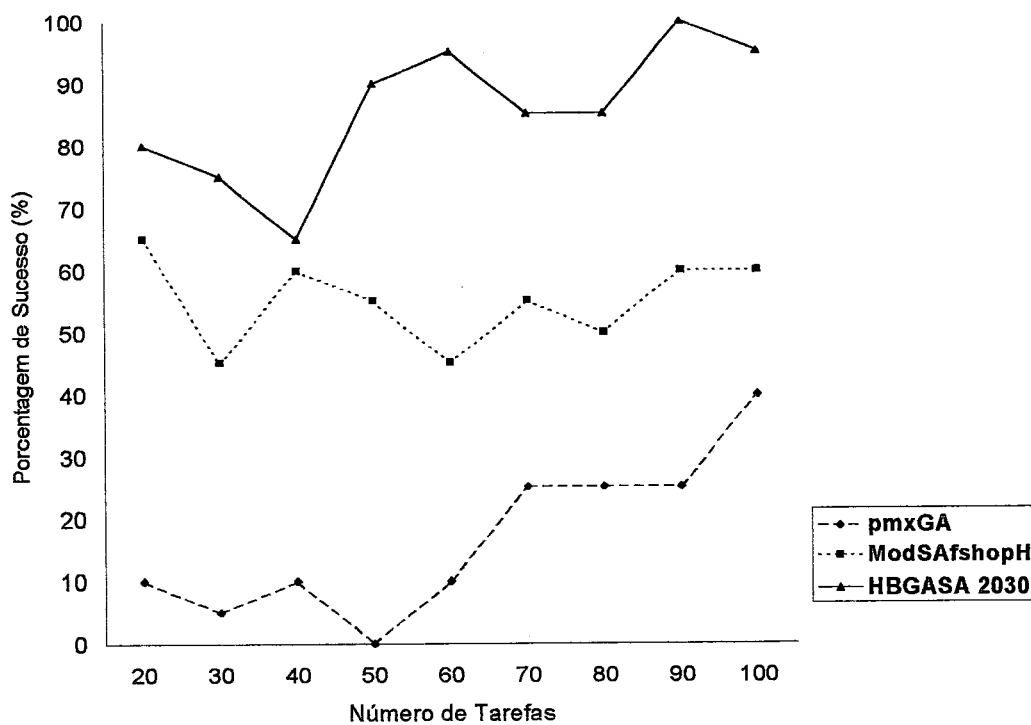


FIGURA 8: Percentagens de Sucesso para máquinas não-agrupadas (m = 7)

pmxGA X ModSAfshopH X HBGASA 2030

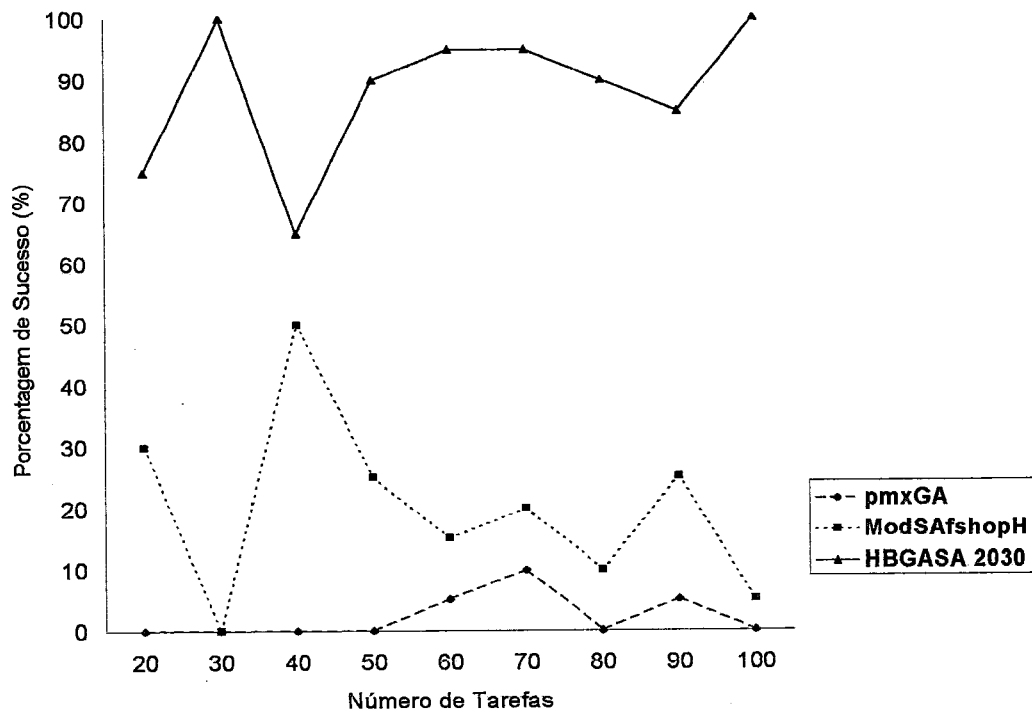


FIGURA 9: Porcentagens de Sucesso para máquinas não-agrupadas ($m = 10$)
 pmxGA X ModSAfshopH X HBGASA 2030

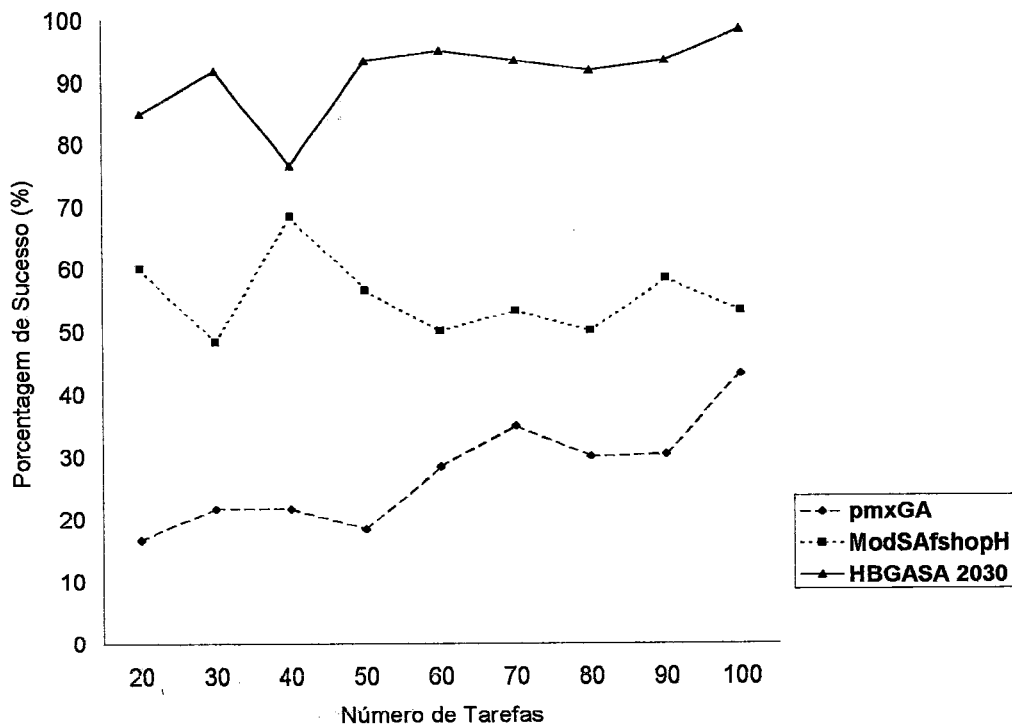


FIGURA 10: Porcentagens de Sucesso para máquinas agrupadas
 pmxGA X ModSAfshopH X HBGASA 2030

O bom desempenho do método *HBGASA 2030*, com operador de cruzamento *PMX*, pode ser observado também através das Melhorias Relativas, conforme mostram as Figuras 11 a 14.

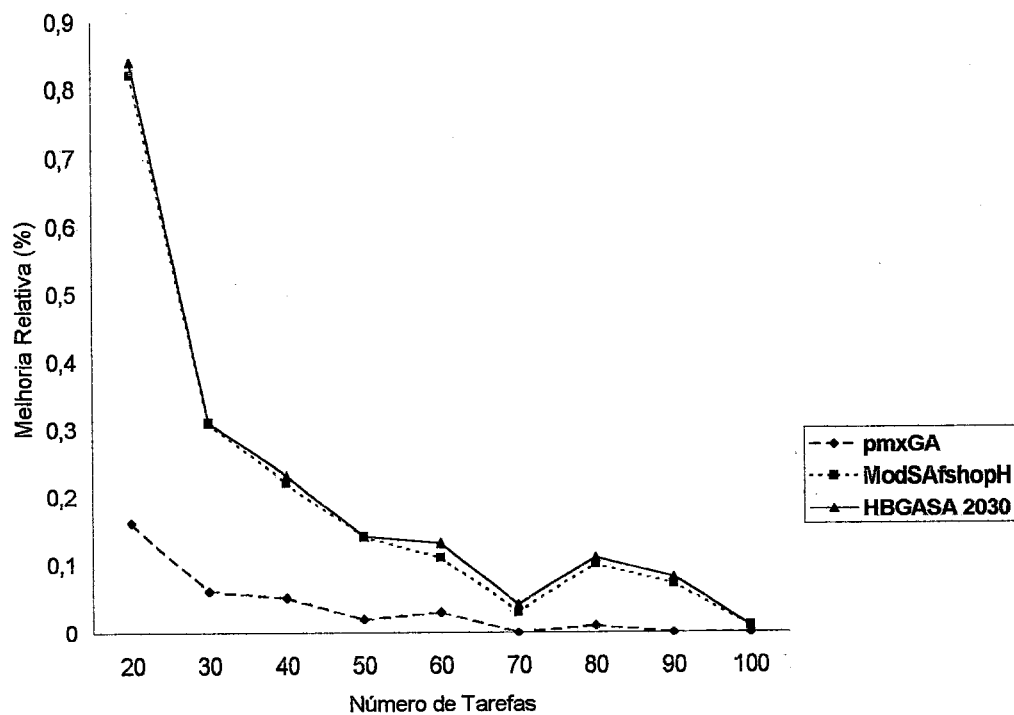


FIGURA 11: Melhorias Relativas para máquinas não-agrupadas ($m = 4$)

pmxGA X ModSAfshopH X *HBGASA 2030*

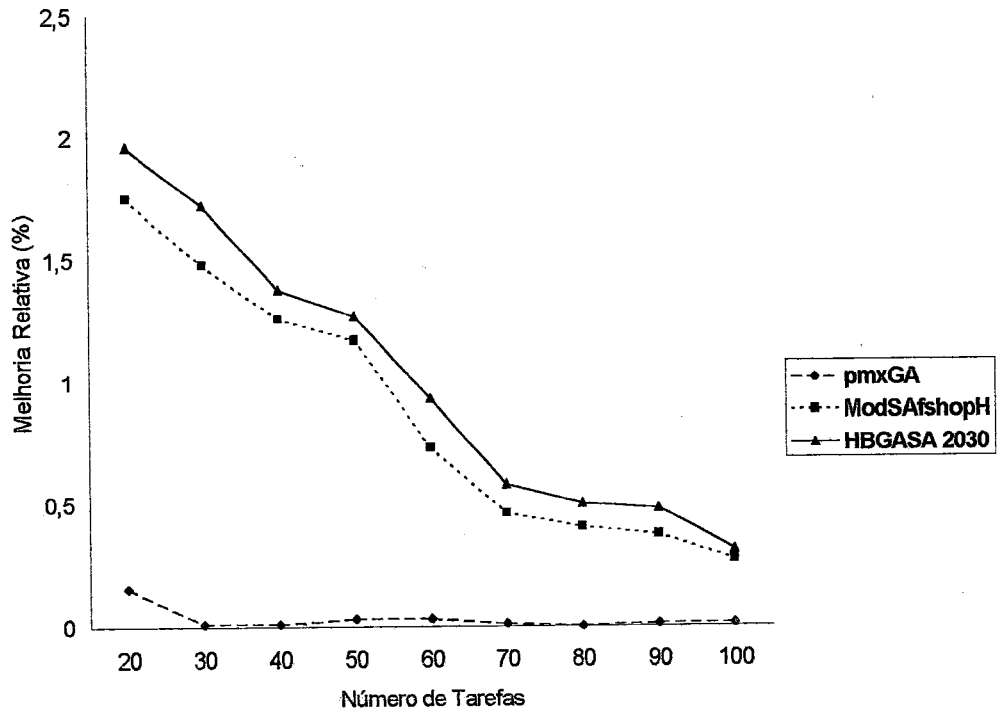


FIGURA 12: Melhorias Relativas para máquinas não-agrupadas (m = 7)

pmxGA X ModSAfshopH X HBGASA 2030

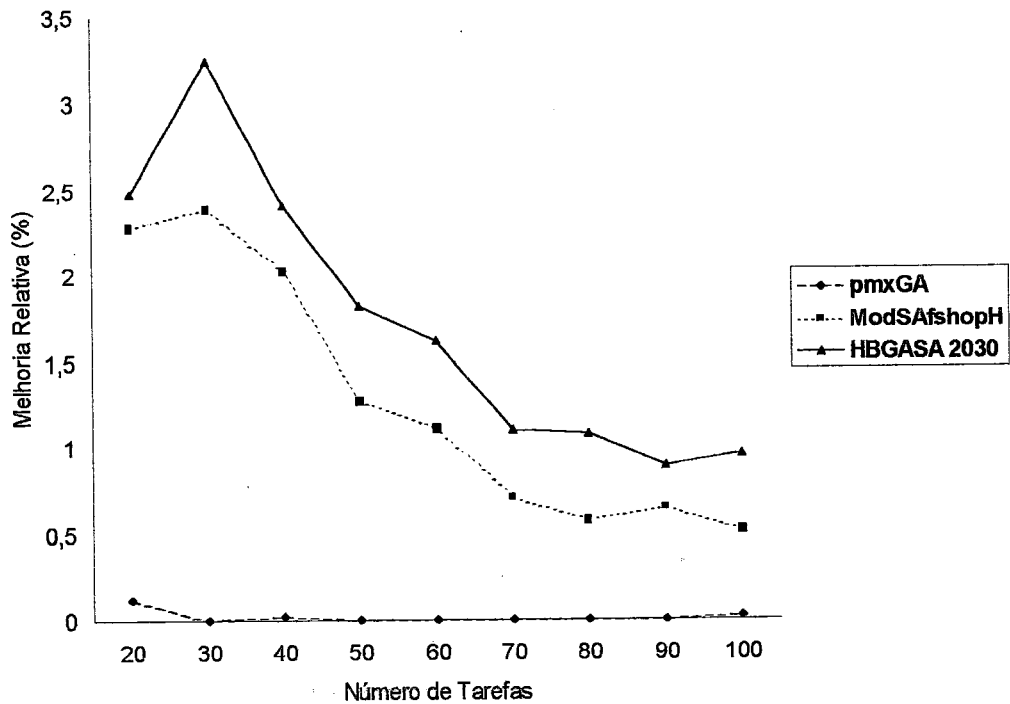


FIGURA 13: Melhorias Relativas para máquinas não-agrupadas (m = 10)

pmxGA X ModSAfshopH X HBGASA 2030

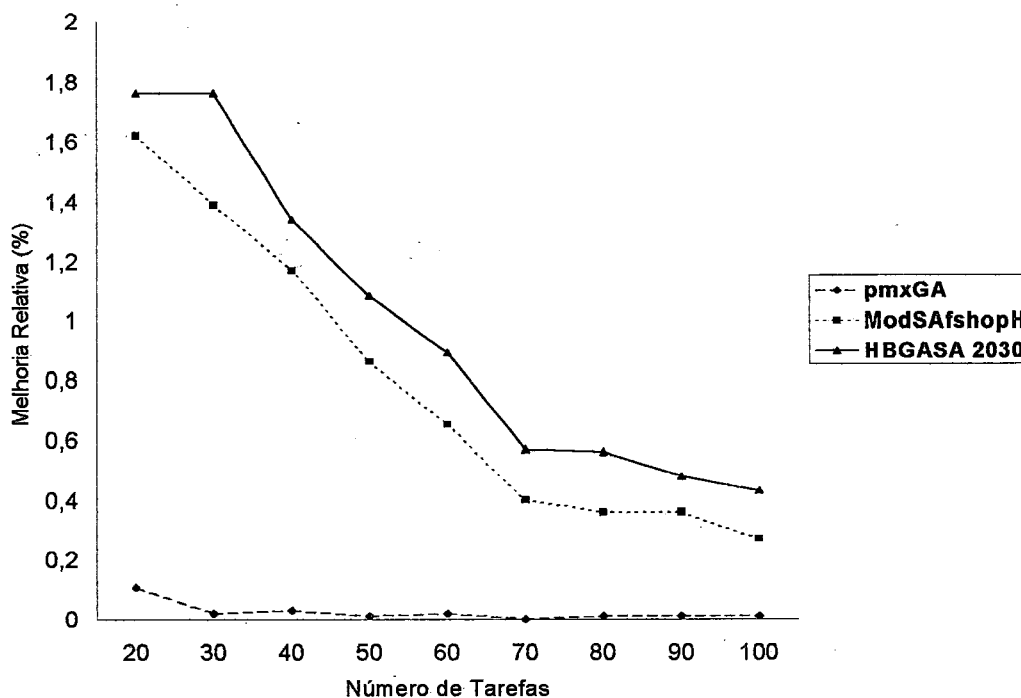


FIGURA 14: Melhorias Relativas para máquinas agrupadas
pmxGA X ModSAfshopH X HBGASA 2030

A Figura 11 mostra que os métodos *HBGASA 2030* e *ModSAfshopH* apresentaram comportamentos praticamente idênticos em relação às Melhorias Relativas, para $m = 4$. Em relação a $m = 7$, ambos os métodos apresentaram comportamentos parecidos, porém o *HBGASA 2030* foi sempre superior quando comparado ao *ModSAfshopH* (Figura 12). No caso de $m = 10$, o *HBGASA 2030* também apresentou desempenho superior quando comparado ao *ModSAfshopH* (Figura 13). Em todos os casos, ambos os métodos superaram o desempenho do *pmxGA*.

Outro resultado estatístico importante no contexto do presente trabalho, diz respeito ao Índice de Desempenho da Busca. As Figuras 15 a 18 ilustram tais resultados.

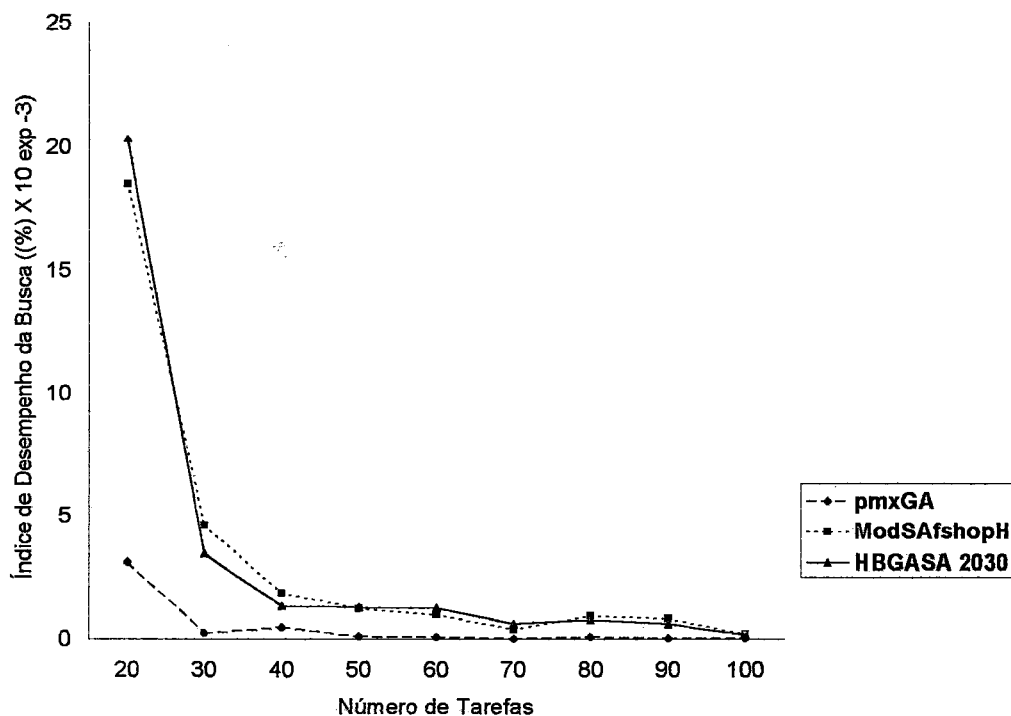


FIGURA 15: Índice de Desempenho da Busca para máquinas não-agrupadas (m = 4)
 pmxGA X ModSAfshopH X HBGASA 2030

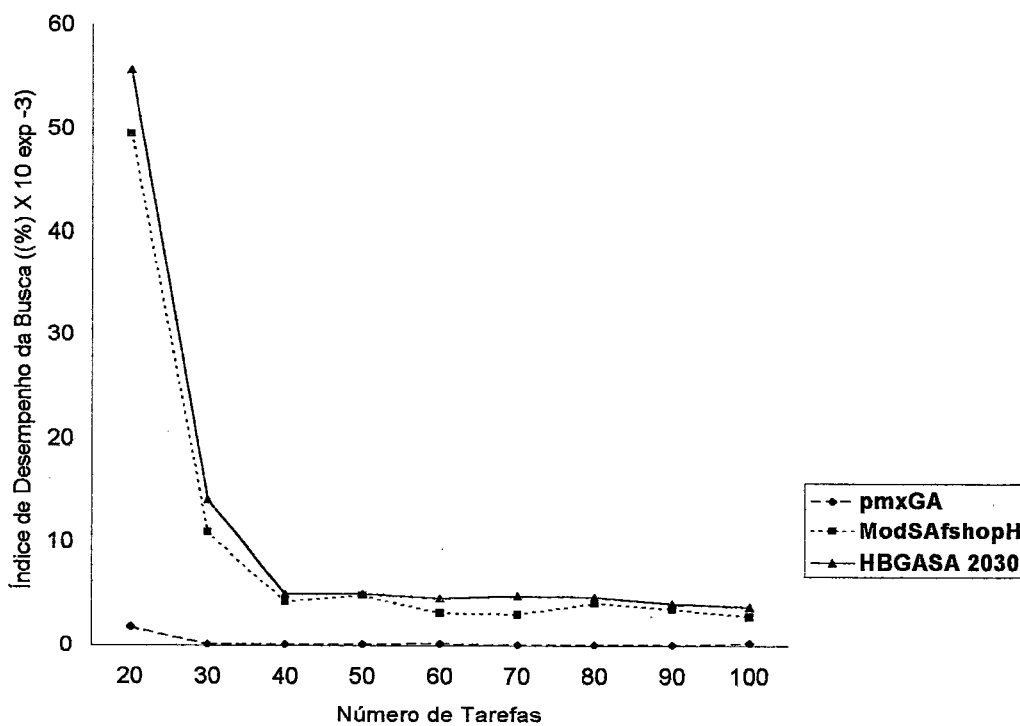


FIGURA 16: Índice de Desempenho da Busca para máquinas não-agrupadas (m = 7)
 pmxGA X ModSAfshopH X HBGASA 2030

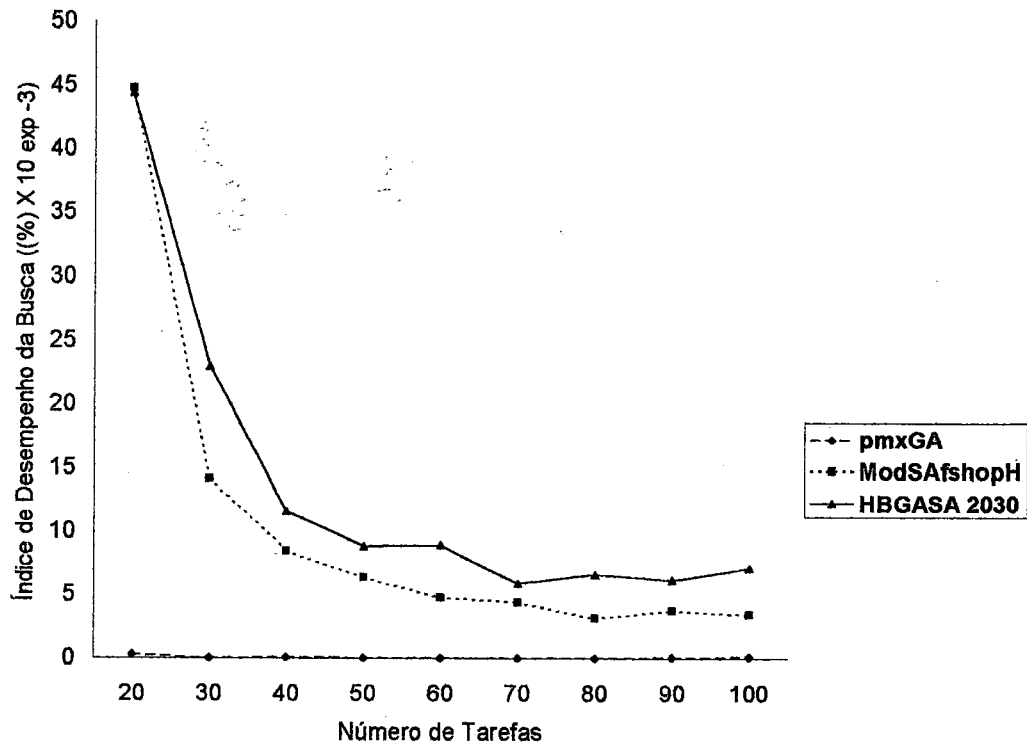


FIGURA 17: Índice de Desempenho da Busca para máqs. não-agrupadas (m = 10)

pmxGA X ModSAfshopH X HBGASA 2030

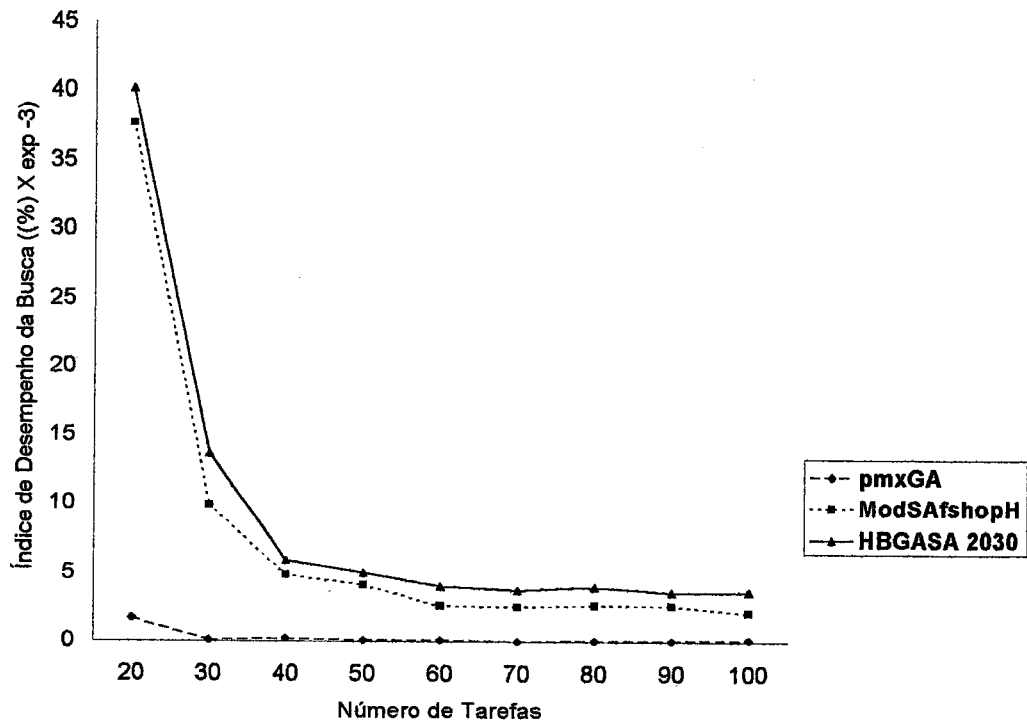


FIGURA 18: Índice de Desempenho da Busca para máquinas agrupadas

pmxGA X ModSAfshopH X HBGASA 2030

Pode-se notar através das Figuras 15 a 18 que, em relação ao Índice de Desempenho da Busca, o método *HBGASA 2030* apresentou comportamento semelhante ao *ModSAfshopH*, para todos os casos de máquinas não-agrupadas. Cabe ressaltar, porém, que o desempenho do método *HBGASA 2030* foi superior ao *ModSAfshopH*. Uma vez mais, ambos os métodos superaram o *pmxGA*.

Ainda em relação aos resultados estatísticos, os três métodos foram comparados em termos do esforço computacional. Tais resultados encontram-se representados nas Figuras 19 a 22.

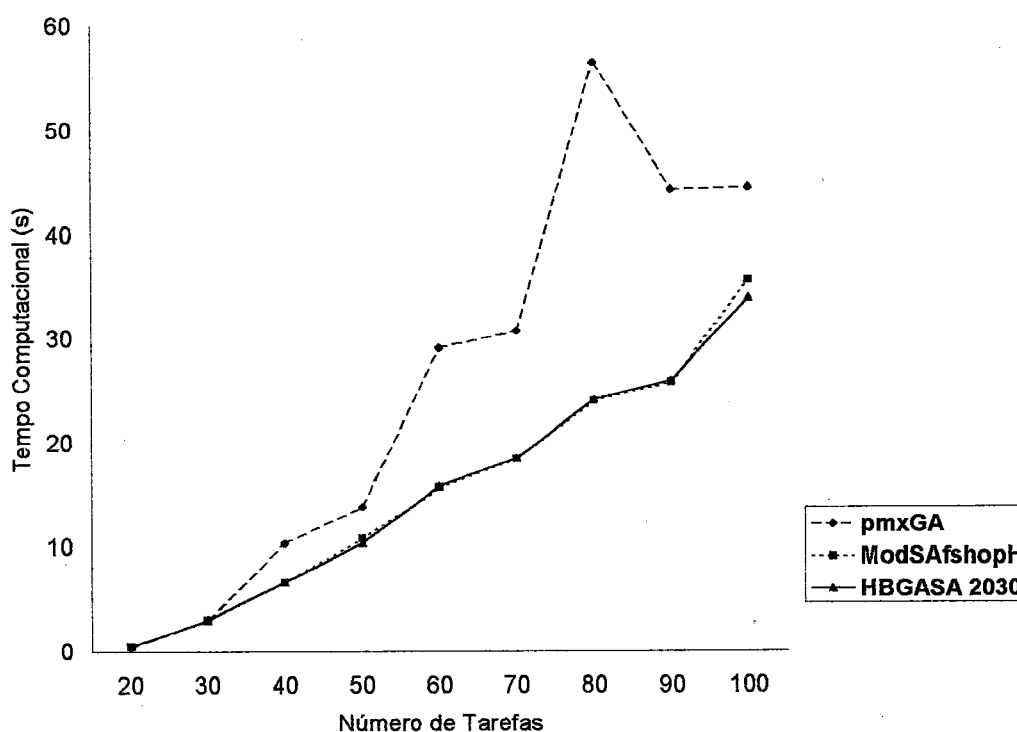


FIGURA 19: Tempos de Computação para máquinas não-agrupadas ($m = 4$)

pmxGA X *ModSAfshopH* X *HBGASA 2030*

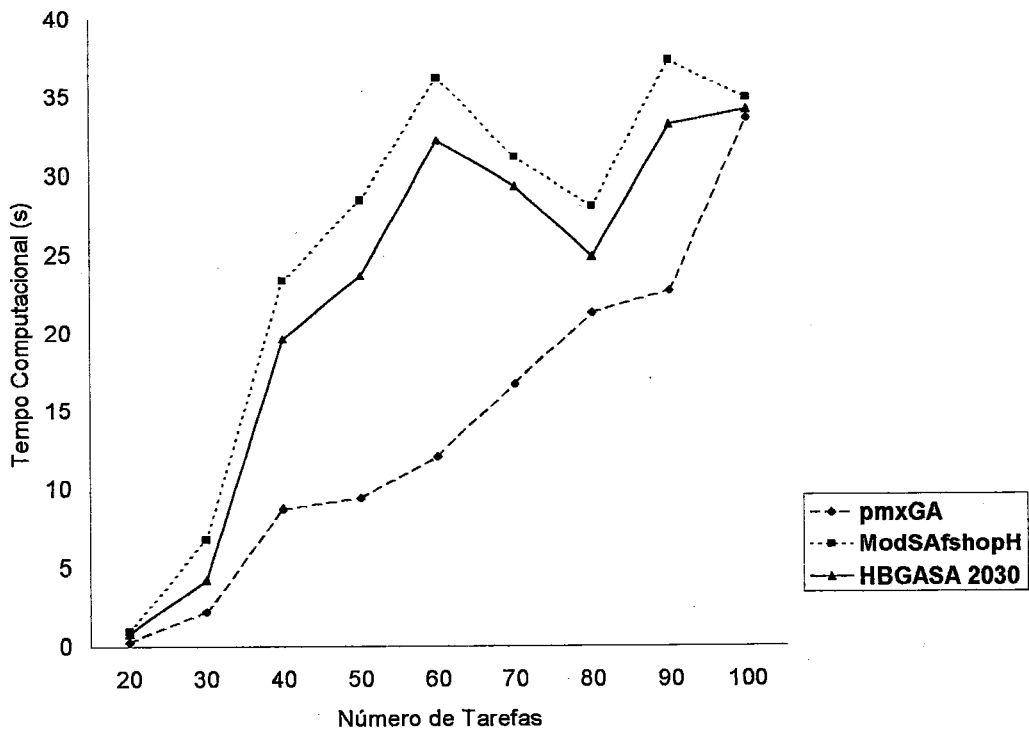


FIGURA 20: Tempos de Computação para máquinas não-agrupadas (m = 7)
pmxGA X ModSAfshopH X HBGASA 2030

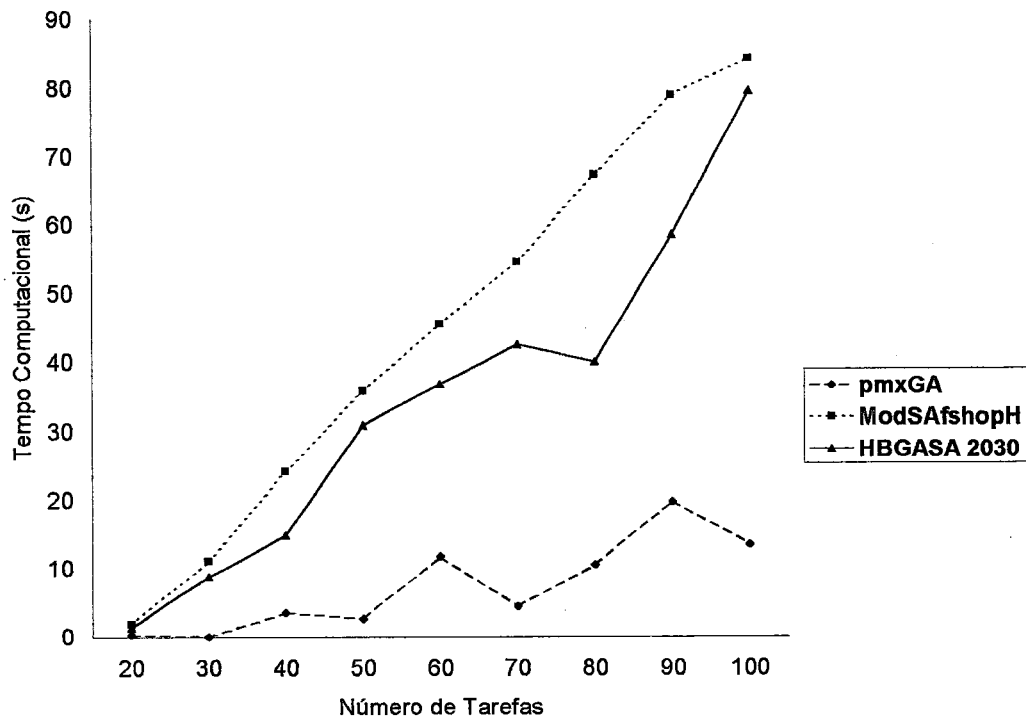


FIGURA 21: Tempos de Computação para máquinas não-agrupadas (m = 10)
pmxGA X ModSAfshopH X HBGASA 2030

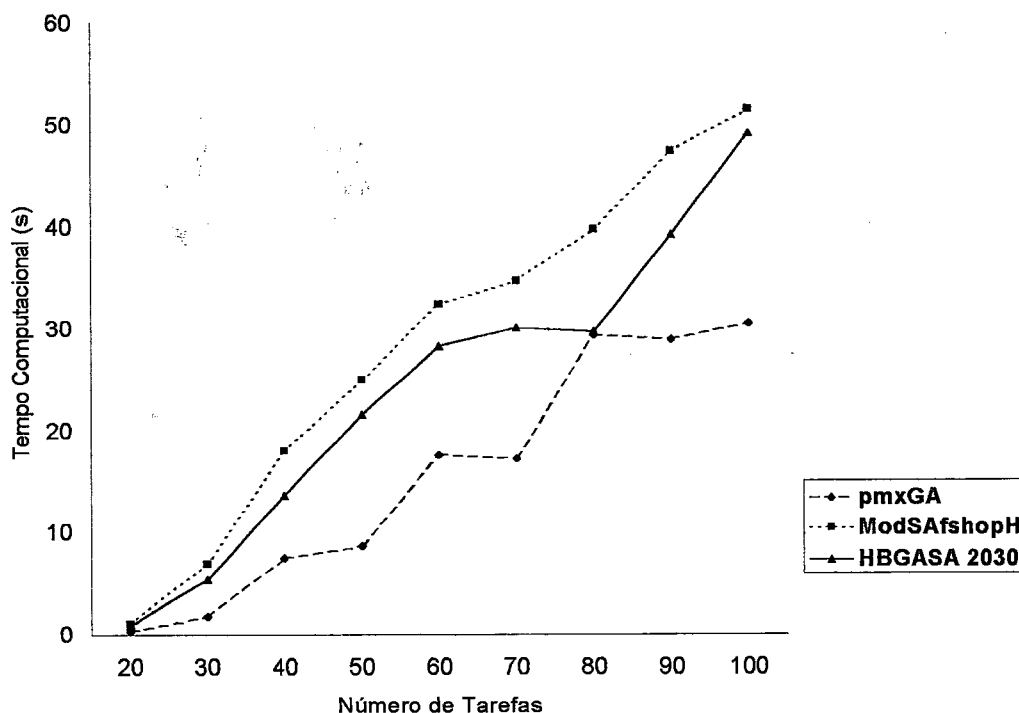


FIGURA 22: Tempos de Computação para máquinas agrupadas
pmxGA X ModSAfshopH X HBGASA 2030

A Figura 19 mostra que os métodos *HBGASA 2030* e *ModSAfshopH* apresentaram padrões de comportamento quase idênticos em relação ao Tempo de Computação, para $m = 4$. O método *pmxGA*, por sua vez, seguiu um padrão de comportamento diferente, quando comparado aos dois outros métodos. Pode-se observar ainda, através da figura em questão, que o método *pmxGA* exigiu uma alta carga computacional na classe de problema: $m = 4$ máquinas e $n = 80$ tarefas. É importante lembrar que o método puro *pmxGA* (bem como o método híbrido *HBGASA*) pode encerrar a busca por melhores soluções quando as seqüências sobre as quais for aplicado o operador de cruzamento forem iguais (é a condição de parada secundária). No caso específico ($m = 4$ e $n = 80$), as seqüências iniciais foram muito diferentes entre si e, portanto, as seqüências-filhos geradas pelo cruzamento também foram totalmente distintas. Conseqüentemente, um número elevado de seqüências foi gerado e analisado pelo *pmxGA*. Devido às características de funcionamento deste método (a aplicação dos operadores de cruzamento e de mutação demanda uma carga computacional considerável), um elevado esforço computacional foi exigido.

Em relação a $m = 7$, apesar do comportamento semelhante, o *HBGASA 2030* exigiu uma carga computacional menor, quando comparado ao *ModSAfshopH* (Figura 20). A Figura 21 mostra que o método *ModSAfshopH* exigiu tempo de computação maior do que o método híbrido, para $m = 10$.

Em suma, as Figuras 19 a 21 mostram que a carga computacional demandada pelo *pmxGA* segue um padrão de comportamento totalmente distinto, quando comparado aos dois outros métodos. Tal diferença é explicada devido às particularidades de funcionamento do Algoritmo Genético puro quanto à “condição de parada”.

Os resultados obtidos na experimentação computacional para os métodos *pmxGA*, *ModSAfshopH* e *HBGASA 2030* (com operador de cruzamento *PMX*) relativos a Porcentagens de Sucesso, Melhorias Relativas, Índices de Desempenho da Busca e Tempos de Computação, para máquinas não-agrupadas e máquinas agrupadas, encontram-se tabelados no **Anexo 1**.

O desenvolvimento dos métodos híbridos, bem como a experimentação computacional, foram conduzidos tomando-se um conjunto de parcelas da vizinhança analisada (p) e um conjunto de temperatura inicial (T_1), ambos representativos e pré-fixados.

Em função da análise dos resultados obtidos, chegou-se à conclusão de que o melhor método híbrido (quando comparado aos metaheurísticos puros) é aquele que opera em $T_1 = 20$ e $p = 30\%$ (operador de cruzamento *PMX*).

Contudo, pode ocorrer uma situação em que a temperatura inicial do método seja diferente daquelas do conjunto pré-fixado no presente trabalho. Neste caso, deve-se analisar em qual parcela da vizinhança analisada o híbrido apresentou o melhor desempenho, para uma T_1 genérica.

Analisando-se os resultados encontrados nas Tabelas 10 a 14 e tomando-se apenas o número de vitórias somado ao número de empates do método híbrido *HBGASA* sobre o *ModSAfshopH*, em relação às Porcentagens de Sucesso (lembrando que $n = 20$ a 100), tem-se o seguinte panorama, conforme a Tabela 17.

TABELA 17: Número de (vitórias + empates) do método HBGASA sobre o ModSAfshopH

$T_1 \backslash p$	05	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	
20	7	8	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
30	8	8	9	8	9	9	9	9	9	9	9	9	9	9	9	8	9	8	9	8	8
45	5	6	8	9	9	9	8	9	7	8	9	8	8	6	5	6	6	6	3	3	3
60	6	5	8	6	7	9	8	7	7	5	6	5	2	4	5	4	2	2	4	2	2
70	6	6	7	8	7	8	6	6	7	4	3	4	2	2	3	1	1	3	1	3	3
TOTAL	32	33	40	40	41	44	40	40	39	35	36	35	30	30	31	28	27	28	26	24	24

A Tabela 17 mostra que, agregando-se os resultados obtidos nas diferentes T_1 , o método *HBGASA* apresentou melhor desempenho frente ao ModSAfshopH (maior número de vitórias e empates) quando $p = 30\%$, confirmando a superioridade dos métodos híbridos **HBGASA T_1 30**.

CAPÍTULO 6

CONCLUSÕES

O presente trabalho teve como finalidade a obtenção de um método metaheurístico híbrido para o problema de Programação de Operações *Flow Shop* Permutacional, com o objetivo de minimizar o *makespan*. Tal método apresenta as características básicas do Algoritmo Genético simples (seleção dos pais e aplicação do operador de cruzamento), porém com um importante diferencial: a diversidade genética (que no Algoritmo Genético puro é gerada pelo operador mutação) é obtida através do método *Simulated Annealing*.

Foram desenvolvidos e testados duzentos algoritmos híbridos (denominados *HBGASA*), combinando-se diferentes temperaturas iniciais e parcelas da vizinhança analisada. Na construção dos métodos híbridos, considerou-se também dois operadores de cruzamento distintos: o operador de cruzamento de UM CORTE e o operador de cruzamento PMX. Os algoritmos híbridos foram comparados com Algoritmos Genéticos puros (*onecutGA*/*pmxGA*) e *Simulated Annealing* puro (*ModSAfshopH*), através de ampla experimentação computacional.

A partir dos resultados obtidos na experimentação computacional, algumas conclusões podem ser relatadas:

☞ O método *HBGASA* com operador de cruzamento PMX apresentou desempenho bastante superior quando comparado ao desempenho observado no híbrido *HBGASA* com operador de cruzamento de UM CORTE.

Apesar da sua relativa eficácia e de ser um dos mais conhecidos e utilizados operadores de cruzamento em Algoritmos Genéticos, o operador de UM CORTE, no problema tratado neste trabalho, possui a característica de gerar seqüências-pais idênticas com alta frequência e geralmente após um número pequeno de iterações. Devido a esta peculiaridade, a condição de parada secundária (encerramento da busca quando as seqüências sobre as quais for aplicado o operador de cruzamento forem iguais) imposta ao método híbrido *HBGASA* e ao Algoritmo Genético puro pode ser logo atingida, o que restringe as possibilidades do método em questão de encontrar soluções de melhor qualidade.

Tal fato não ocorre com o operador de cruzamento PMX, uma vez que este operador é capaz de gerar uma maior variedade de seqüências-filhos em relação às seqüências-pais, originalmente diferentes.

☞ O desempenho do método *HBGASA* melhorou em função do aumento do tamanho do problema, isto é, quanto maior o número de tarefas do problema, melhor foi a performance do híbrido, quando comparada aos métodos Algoritmo Genético e *Simulated Annealing* puros. Tal comportamento foi observado em relação aos dois operadores de cruzamento utilizados.

☞ Em relação à parcela da vizinhança analisada (p), observou-se que, de um modo geral, o método *HBGASA* com operador de cruzamento de UM CORTE mostrou melhor desempenho quando trabalhou com valores relativamente altos do parâmetro p . No caso do operador de cruzamento PMX, o método *HBGASA* apresentou melhor desempenho quando utilizou baixos valores de p , com destaque para $p = 20\%$ e $p = 30\%$.

☞ O método *HBGASA* com operador de cruzamento de UM CORTE superou os métodos Algoritmo Genético e *Simulated Annealing* puros somente quando a temperatura inicial foi baixa (mais precisamente, quando $T_1 = 20$). Em $T_1 = 30, 45, 60$ e 70 , o método ModSAfshopH se mostrou bastante superior ao *HBGASA*.

Um comportamento totalmente diferente foi observado em relação ao híbrido *HBGASA* com operador de cruzamento PMX, uma vez que tal método superou os métodos puros nos casos $T_1 = 20, 30, 45$ e 60 , e foi superado pelo ModSAfshopH somente quando $T_1 = 70$ e $p = 0,95$.

☞ O desempenho do Algoritmo Genético puro se mostrou sempre inferior, quando comparado ao ModSAfshopH e ao *HBGASA*. Os métodos onecutGA e pmxGA foram superados pelo ModSAfshopH e/ou *HBGASA* (operador de cruzamento de UM CORTE e PMX) para todos os cálculos estatísticos realizados, em todas as classes de problemas estudadas.

☞ De um modo geral, os tempos computacionais demandados pelos métodos ModSAfshopH e *HBGASA* foram bem próximos entre si. No entanto, este último método demandou uma carga computacional um pouco inferior, quando comparado ao ModSAfshopH. O Algoritmo Genético puro apresentou um comportamento de carga computacional demandada totalmente distinto: em alguns casos, superou os métodos ModSAfshopH e *HBGASA* e, em outras situações, exigiu uma carga computacional inferior, não seguindo, portanto, um comportamento mais estável.

☞ O método híbrido *HBGASA* (com operador de cruzamento PMX) obteve o melhor desempenho quando $p = 30\%$, independentemente do valor de T_1 adotado. Portanto, para T_1 genérica, o valor de p mais apropriado é: 30% .

Através dos resultados obtidos na experimentação computacional, chegou-se à conclusão de que o melhor dentre todos os métodos híbridos desenvolvidos e testados foi o método **HBGASA 2030 (com operador de cruzamento PMX)**. Tal método superou os desempenhos apresentados pelos métodos puros Algoritmo Genético e *Simulated Annealing*, para as estatísticas de Porcentagens de Sucesso, Melhorias Relativas e Índice de Desempenho da Busca, em relação à grande maioria das classes de problemas (m,n) estudadas.

Considerando-se os resultados, observações e conclusões aqui relatados, cabem algumas sugestões para futuros trabalhos nesta linha de pesquisa:

➡ A utilização de operadores de cruzamento diferentes daqueles empregados no presente trabalho. Tais operadores podem ser encontrados na literatura.

➡ O emprego de um operador de cruzamento “inteligente”, que utilize as características genéticas favoráveis (posições relativas das tarefas) nos cromossomos (seqüências) pais.

➡ A utilização de diferentes funções de resfriamento e diferentes probabilidades de aceitação, no procedimento *Simulated Annealing*.

ANEXO 1

Resultados dos Problemas do Experimento

**(métodos pmxGA, ModSAfshopH e *HBGASA* 2030 –
operador de cruzamento PMX)**

Os resultados estatísticos relativos aos métodos pmxGA, ModSAfshopH e HBGASA 2030 (operador de cruzamento PMX) para máquinas não-agrupadas estão relatados nas Tabelas 18 a 21.

TABELA 18: Porcentagens de Sucesso da classe (%) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas não-agrupadas)

Algoritmo	Nro. de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	40,00	60,00	55,00	55,00	69,99	69,99	64,99	60,00	89,99
	7	10,00	5,00	10,00	0	10,00	25,00	25,00	25,00	40,00
	10	0	0	0	0	5,00	10,00	0	5,00	0
ModSAfshopH	4	85,00	100,0	94,99	89,99	89,99	85,00	89,99	89,99	94,99
	7	64,99	44,99	60,00	55,00	44,99	55,00	50,00	60,00	60,00
	10	30,00	0	50,00	25,00	15,00	20,00	10,00	25,00	5,00
HBGASA 2030	4	100,0	100,0	100,0	100,0	94,99	100,0	100,0	94,99	100,0
	7	80,00	75,00	64,99	89,99	94,99	85,00	85,00	100,0	94,99
	10	75,00	100,0	64,99	89,99	94,99	94,99	89,99	85,00	100,0

TABELA 19: Melhorias Relativas da classe (%) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas não-agrupadas)

Algoritmo	Nro. de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	0,16	0,06	0,05	0,02	0,03	0	0,01	0	0
	7	0,15	0,01	0,01	0,03	0,03	0,01	0	0,01	0,01
	10	0,12	0	0,02	0	0	0	0	0	0,02
ModSAfshopH	4	0,82	0,31	0,22	0,14	0,11	0,03	0,10	0,07	0,01
	7	1,75	1,48	1,26	1,17	0,73	0,46	0,40	0,37	0,27
	10	2,28	2,39	2,03	1,27	1,11	0,71	0,58	0,65	0,52
HBGASA 2030	4	0,84	0,31	0,23	0,14	0,13	0,04	0,11	0,08	0,01
	7	1,96	1,72	1,38	1,27	0,93	0,58	0,50	0,48	0,31
	10	2,48	3,25	2,41	1,83	1,62	1,10	1,08	0,89	0,96

TABELA 20: Índices de Desempenho da Busca da classe (em porcentagem $\times 10^{-3}$) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas não-agrupadas)

Algoritmo	Nro. De Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	3,09	0,22	0,45	0,09	0,07	0	0,06	0,03	0,02
	7	1,84	0,08	0,09	0,12	0,14	0,03	0,04	0,04	0,23
	10	0,29	0	0,07	0,03	0,03	0	0	0,05	0,10
ModSAfshopH	4	18,56	4,57	1,80	1,18	0,93	0,36	0,87	0,77	0,18
	7	49,48	10,89	4,15	4,76	3,08	2,99	4,00	3,49	2,82
	10	44,77	14,14	8,51	6,38	4,75	4,40	3,22	3,80	3,48
HBGASA 2030	4	20,30	3,40	1,28	1,22	1,19	0,58	0,72	0,58	0,15
	7	55,68	14,15	4,91	4,89	4,48	4,71	4,58	4,00	3,64
	10	44,42	23,03	11,56	8,84	8,95	5,85	6,61	6,14	7,12

TABELA 21: Tempos Médios de Computação da classe (em segundos) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas não-agrupadas)

Algoritmo	Nro. de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	0,42	3,03	10,26	13,80	29,02	30,63	56,32	44,21	44,42
	7	0,27	2,14	8,73	9,37	12,08	16,69	21,21	22,6	33,57
	10	0,22	0,02	3,41	2,58	11,65	4,25	10,45	19,72	13,30
ModSAfshopH	4	0,50	2,95	6,64	10,71	15,59	18,34	23,89	25,74	35,39
	7	0,92	6,76	23,30	28,46	36,18	31,17	28,06	37,32	34,86
	10	1,80	11,00	24,03	35,73	45,42	54,48	67,22	78,70	84,08
HBGASA 2030	4	0,51	3,00	6,64	10,29	15,75	18,39	24,20	25,86	33,80
	7	0,76	4,25	19,58	23,60	32,20	29,31	24,87	33,22	34,11
	10	1,27	8,69	14,67	30,74	36,65	42,32	39,88	58,44	79,26

Os resultados estatísticos relativos aos métodos pmxGA, ModSAfshopH e HBGASA 2030 (operador de cruzamento PMX) para máquinas agrupadas estão relatados nas Tabelas 22 a 25.

TABELA 22: Porcentagens de Sucesso (%) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas agrupadas)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
pmxGA	16,66	21,66	21,66	18,33	28,33	34,99	29,99	30,33	43,33
ModSAfshopH	60,00	48,33	68,33	56,66	50,00	53,33	50,00	58,33	53,33
HBGASA 2030	85,00	91,66	76,66	93,33	94,99	93,33	91,66	93,33	98,33

TABELA 23: Melhorias Relativas (%) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas agrupadas)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
pmxGA	0,11	0,02	0,03	0,01	0,02	0	0,01	0,01	0,01
ModSAfshopH	1,62	1,39	1,17	0,86	0,65	0,40	0,36	0,36	0,27
HBGASA 2030	1,76	1,76	1,34	1,08	0,89	0,57	0,56	0,48	0,43

TABELA 24: Índices de Desempenho da Busca (em porcentagem $\times 10^{-3}$) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas agrupadas)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
pmxGA	1,74	0,10	0,20	0,08	0,06	0,01	0,03	0,04	0,12
ModSAfshopH	37,60	9,87	4,82	4,11	2,65	2,58	2,70	2,68	2,16
HBGASA 2030	40,16	13,53	5,92	4,98	4,01	3,71	3,97	3,57	3,64

TABELA 25: Tempos Médios de Computação (em segundos) para pmxGA, ModSAfshopH e HBGASA 2030 (máquinas agrupadas)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
pmxGA	0,31	1,73	7,47	8,59	17,58	17,19	29,33	28,85	30,43
ModSAfshopH	1,07	6,90	17,99	24,97	32,40	34,66	39,72	47,25	51,44
HBGASA 2030	0,85	5,31	13,63	21,54	28,20	30,01	29,65	39,17	49,02



ANEXO 2

Programas Computacionais

IMPLEMENTAÇÃO COMPUTACIONAL

VARIÁVEIS GLOBAIS

```

VAR
  i, j          : INTEGER;
  TUBgap        : INTEGER;
  DstartS1, DstartS2 : INTEGER;
  StartS        : StartSEQS;
  UBG           : ARRNN;
  BestInitS     : ARRNN;
  DBestInitS    : INTEGER;
  h,min,snd,hund : WORD;

  T1, TK        : REAL;
  B, Temp       : REAL;
  Percent, INCpercent : REAL;
  startTime     : REAL;
  InitSsTime1, InitSsTime2 : REAL;
  HeurSndTime   : CPUTIMES;

  equalJobs     : INTEGER;
  SameSeeds     : BOOLEAN;

```

PROCEDIMENTOS E FUNÇÕES AUXILIARES

```

PROCEDURE UBGij(
  VAR m,n : INTEGER;
  VAR P : ARRNN;
  VAR UBG : ARRNN );

VAR
  k,i,j : INTEGER;
  DK : ARRNN;

PROCEDURE DKij(
  VAR k : INTEGER;
  VAR P : ARRNN;
  VAR DK : ARRNN);

VAR
  i,j : INTEGER;

BEGIN
  (* DKij *)
  FOR i:=1 TO n DO
    FOR j:=1 TO n DO
      IF i=j THEN DK[i,j] := INF
      ELSE DK[i,j] := P[k,j] - P[k+1,i]
    END;
  END;
  (* DKij *)

BEGIN
  (* UBGij *)
  k := 1 ;
  DKij(k,P,DK);

```



```

PROCEDURE HEURISTICtime(
  VAR secondsTime1, secondsTime2 : REAL;
  VAR Htime : REAL);

BEGIN
  (* HEURISTICtime *)
  Htime := 0.0;

  IF secondsTime2 < secondsTime1 THEN
    Htime := 86400.0 + secondsTime2 - secondsTime1
  ELSE
    Htime := secondsTime2 - secondsTime1;

END;
  (* HEURISTICtime *)

FUNCTION NextTemp(T, b : REAL) : REAL;

  VAR
    Ntemp : REAL;

  BEGIN
    (* NextTemp *)
    Ntemp := 0.0;

    Ntemp := T / (1.0 + b*T);
    IF Ntemp < 1.0 THEN Ntemp := 1.0;
    NextTemp := Ntemp;

  END;
    (* NextTemp *)

FUNCTION NbTS5Neighbours(VAR N : INTEGER) : INTEGER;

  VAR AUX1, AUX2 : REAL;
  VAR IntAUX2 : INTEGER;

  BEGIN
    (* NbTS5Neighbours *)

    NbTS5Neighbours := 0; AUX1 := 0.0 ; AUX2 := 0.0;
    IntAUX2 := 0;

    IF N <= 31 THEN
      NbTS5Neighbours := (N-2)*(N-2)

    ELSE
      BEGIN
        AUX1 := 0.0 - 0.04*(N-31);
        AUX2 := 900.0*Exp(AUX1);
        IntAUX2 := TRUNC(AUX2);
        NbTS5Neighbours := 1741 - IntAUX2
      END;

  END;
    (* NbTS5Neighbours *)

FUNCTION NbHybridSasequences(
  VAR n : INTEGER;
  VAR coef : REAL ) : INTEGER;

  VAR AUX : REAL;

```

```

BEGIN                                (* NbHybridSAsequences *)

    NbHybridSAsequences := 0; AUX := 0.0;
    AUX := coef * NbTS5Neighbours(n);
    NbHybridSAsequences := ROUND(AUX);

END;                                (* NbHybridSAsequences *)

PROCEDURE NbSequencesGAandModSAandHybrid(
    VAR m, n      : INTEGER;
    VAR NbSequences : LONGINT);

BEGIN                                { * NbSequencesGAandModSAandHybrid * }
    NbSequences := 0 ;

    IF (m=4) THEN
        BEGIN
            IF (n=21) THEN NbSequences := 9693;
            IF (n=31) THEN NbSequences := 43732;
            IF (n=41) THEN NbSequences := 77742;
            IF (n=51) THEN NbSequences := 105623;
            IF (n=61) THEN NbSequences := 133502;
            IF (n=71) THEN NbSequences := 137452;
            IF (n=81) THEN NbSequences := 166050;
            IF (n=91) THEN NbSequences := 155455;
            IF (n=101) THEN NbSequences := 194680;
        END;

    IF (m=7) THEN
        BEGIN
            IF (n=21) THEN NbSequences := 10812;
            IF (n=31) THEN NbSequences := 59666;
            IF (n=41) THEN NbSequences := 162358;
            IF (n=51) THEN NbSequences := 165856;
            IF (n=61) THEN NbSequences := 178017;
            IF (n=71) THEN NbSequences := 135330;
            IF (n=81) THEN NbSequences := 107244;
            IF (n=91) THEN NbSequences := 128733;
            IF (n=101) THEN NbSequences := 108136;
        END;

    IF (m=10) THEN
        BEGIN
            IF (n=21) THEN NbSequences := 16750;
            IF (n=31) THEN NbSequences := 73503;
            IF (n=41) THEN NbSequences := 125728;
            IF (n=51) THEN NbSequences := 154223;
            IF (n=61) THEN NbSequences := 166257;
            IF (n=71) THEN NbSequences := 173862;
            IF (n=81) THEN NbSequences := 189378;
            IF (n=91) THEN NbSequences := 189406;
            IF (n=101) THEN NbSequences := 190884;
        END;

END;                                { * NbSequencesGAandModSAandHybrid * }

```

```

PROCEDURE GetRandomSAShiftNeighbour(
  VAR s, NB : ARR);

VAR
  h, i, j, aux : INTEGER;

BEGIN          (* GetRandomSAShiftNeighbour *)

  RANDOMIZE;
  h := 0 ; i := 0 ; aux := 0 ;

  REPEAT
    h := Random(n-1);
    h := h + 2;
    i := Random(n-1);
    i := i + 2;
  UNTIL h <> i;

  IF h < i THEN
    BEGIN
      aux := s[h];
      FOR j := h TO (i-1) DO
        begin
          s[j] := s[j+1];
        end;
      s[i] := aux;
    END
  ELSE
    BEGIN
      aux := s[h];
      FOR j := h DOWNTO (i+1) DO
        begin
          s[j] := s[j-1];
        end;
      s[i] := aux;
    END;
  NB := s;

END;          (* GetRandomSAShiftNeighbour *)

```

```

PROCEDURE GetSACurrentSequence(
  VAR Seq, NBSeq, NewSeq : ARR;
  VAR Temperature : REAL;
  VAR DSeq, DNBSSeq, DNewSeq: INTEGER);

VAR
  d, R, IntProb : INTEGER;
  Prob, E : REAL;

BEGIN          (* GetSACurrentSequence *)

  d := DNBSSeq - DSeq;

  IF d <= 0 THEN
    BEGIN
      NewSeq := NBSeq;
      DNewSeq := DNBSSeq;

```



```

END
ELSE
BEGIN
  E := 0 - (d / Temperature);
  Prob := Exp(E);
  Prob := 100*Prob;
  IntProb := ROUND(Prob);

  RANDOMIZE;
  R := 0 ;
  R := Random(101);

  IF R <= IntProb THEN
  BEGIN
    NewSeq := NBSeq;
    DNewSeq := DNBSeg;
  END
  ELSE
  BEGIN
    NewSeq := Seq;
    DNewSeq := DSeq;
  END;
END;
END;          (* GetSACurrentSequence *)

PROCEDURE onecutOPERATOR(
  VAR n : INTEGER;
  VAR s1, s2 : ARR;
  VAR s3, s4 : ARR);

VAR
  i, j, CutPosition : INTEGER ;
  u, v, vaux      : INTEGER;
  NbCrossATTEMPT  : INTEGER;
  AUXs3, AUXs4    : ARR;
  STOP, CROSSOVER : BOOLEAN;

BEGIN          (* onecutOPERATOR *)

  CROSSOVER := FALSE;

  NbCrossATTEMPT := 0;

  RANDOMIZE;

  REPEAT
    CutPosition := 0 ;

    REPEAT
      CutPosition := Random(n) ;
    UNTIL CutPosition > 3 ;

    FOR i := 1 TO CutPosition DO
      BEGIN
        AUXs3[i] := s1[i] ;
        AUXs4[i] := s2[i] ;
      END;
    END;
  END;

```

```

FOR i := (CutPosition + 1) TO n DO
  BEGIN
    AUXs3[i] := s2[i] ;
    AUXs4[i] := s1[i] ;
  END;

{ viabilizacao das sequencias AUXs3 e AUXs4 }

FOR i := 2 TO CutPosition DO
  FOR j := (CutPosition+1) TO n DO
    BEGIN
      IF AUXs3[i] = AUXs3[j] THEN
        BEGIN
          u := 1;
          STOP := FALSE ;

          REPEAT
            u := u + 1 ; v := CutPosition ;

            REPEAT
              v := v + 1 ;
              IF AUXs4[u] = AUXs4[v] THEN
                BEGIN
                  vaux := AUXs3[i] ;
                  AUXs3[i] := AUXs4[u] ;
                  AUXs4[u] := vaux ;
                  STOP := TRUE ;
                END;
            UNTIL (STOP) OR (v = n) ;

          UNTIL (STOP) OR (u = CutPosition) ;
        END;
      END;
    END;

  i := 0 ;

  REPEAT
    i := i + 1 ;
    IF AUXs3[i] = s2[i] THEN CROSSOVER := FALSE
    ELSE CROSSOVER := TRUE ;

  UNTIL (CROSSOVER) OR (i = CutPosition) ;

  NbCrossATTEMPT := NbCrossATTEMPT + 1 ;
  IF NbCrossATTEMPT = 3*(n-1) THEN CROSSOVER := TRUE ;

  UNTIL CROSSOVER ;

{ obtencao das sequencias s3 e s4 viabilizadas }

FOR i := 1 TO n DO
  BEGIN
    s3[i] := AUXs3[i] ;
    s4[i] := AUXs4[i] ;
  END;

END;          (* onecutOPERATOR *)

```

```

PROCEDURE pmxOPERATOR(
  VAR n : INTEGER;
  VAR s1, s2 : ARRN;
  VAR s3, s4 : ARRN);

VAR
  i, j, Cut1, Cut2 : INTEGER ;
  range, MAXrange : INTEGER;
  NbCrossATTEMPT : INTEGER;
  AUXs3, AUXs4 : ARRN;
  CROSSOVER, FEASIBLE : BOOLEAN;

BEGIN          (* pmxOPERATOR *)

  CROSSOVER := FALSE;
  FEASIBLE := FALSE;

  NbCrossATTEMPT := 0;
  MAXrange := ROUND( (n-1)/2 );

  RANDOMIZE;

  REPEAT
    Cut1 := 0 ; Cut2 := 0 ; range := 0 ;

    REPEAT
      Cut1 := Random(n-3); Cut1 := Cut1 + 2 ;
      range := Random(MAXrange); range := range + 1 ;
      Cut2 := Cut1 + range ;
    UNTIL Cut2 <= (n-1) ;

    {determinacao das sequencias auxiliares AUXs3 e AUXs4 }
    FOR i := 1 TO Cut1 DO
      BEGIN
        AUXs3[i] := s1[i] ;
        AUXs4[i] := s2[i] ;
      END;

    FOR i := (Cut1+1) TO Cut2 DO
      BEGIN
        AUXs3[i] := s2[i] ;
        AUXs4[i] := s1[i] ;
      END;

    FOR i := (Cut2+1) TO n DO
      BEGIN
        AUXs3[i] := s1[i] ;
        AUXs4[i] := s2[i] ;
      END;

    {AUXs3 e AUXs4 obtidas}

    {viabilizacao de AUXs3 e AUXs4}

    REPEAT
      FOR j := 2 TO Cut1 DO
        FOR i := (Cut1+1) TO Cut2 DO

```

```

BEGIN
  IF AUXs3[i] = AUXs3[j] THEN AUXs3[j] := s1[i] ;
  IF AUXs4[i] = AUXs4[j] THEN AUXs4[j] := s2[i] ;
END;

FOR j := (Cut2+1) TO n DO
  FOR i := (Cut1+1) TO Cut2 DO
    BEGIN
      IF AUXs3[i] = AUXs3[j] THEN AUXs3[j] := s1[i] ;
      IF AUXs4[i] = AUXs4[j] THEN AUXs4[j] := s2[i] ;
    END;

FEASIBLE := TRUE;

FOR j := 2 TO Cut1 DO
  FOR i := (Cut1+1) TO Cut2 DO
    BEGIN
      IF AUXs3[i] = AUXs3[j] THEN FEASIBLE := FALSE ;
      IF AUXs4[i] = AUXs4[j] THEN FEASIBLE := FALSE ;
    END;

FOR j := (Cut2+1) TO n DO
  FOR i := (Cut1+1) TO Cut2 DO
    BEGIN
      IF AUXs3[i] = AUXs3[j] THEN FEASIBLE := FALSE ;
      IF AUXs4[i] = AUXs4[j] THEN FEASIBLE := FALSE ;
    END;

UNTIL FEASIBLE ;

{ AUXs3 e AUXs4 viabilizadas}

{ verificacao se houve CRUZAMENTO }
i := 1 ;
REPEAT
  i := i + 1 ;
  IF AUXs3[i] = s2[i] THEN CROSSOVER := FALSE
  ELSE CROSSOVER := TRUE ;
UNTIL (CROSSOVER) OR ( i = Cut1 );

IF CROSSOVER = FALSE THEN
  BEGIN
    i := Cut2 ;
    REPEAT
      i := i + 1 ;
      IF AUXs3[i] = s2[i] THEN CROSSOVER := FALSE
      ELSE CROSSOVER := TRUE ;
    UNTIL (CROSSOVER) OR ( i = n );
  END;

i := 1 ;
REPEAT
  i := i + 1 ;
  IF AUXs3[i] = s1[i] THEN CROSSOVER := FALSE
  ELSE CROSSOVER := TRUE ;
UNTIL (CROSSOVER) OR ( i = n );

NbCrossATTEMPT := NbCrossATTEMPT + 1 ;

```

```

IF NbCrossATTEMPT = 3*(n-1) THEN CROSSOVER := TRUE ;

UNTIL CROSSOVER ;
{CRUZAMENTO ocorrido ou ACEITO apos 3n tentativas }

{obtencao das sequencias s3 e s4 viabilizadas }

FOR i := 1 TO n DO
  BEGIN
    s3[i] := AUXs3[i] ;
    s4[i] := AUXs4[i] ;
  END;

END;          (* pmxOPERATOR *)

PROCEDURE MUTATION1(
  VAR s3, s4 : ARRn);

VAR
  locus      : INTEGER;
  aux3, aux4 : INTEGER;

BEGIN          (* MUTATION1 *)

  RANDOMIZE;

  locus := random(n-2);
  locus := locus + 2;

  aux3 := s3[locus];
  s3[locus] := s3[locus+1];
  s3[locus+1] := aux3;
  aux4 := s4[locus];
  s4[locus] := s4[locus+1];
  s4[locus+1] := aux4;
END;          (* MUTATION1 *)

PROCEDURE MUTATION2(
  VAR s3, s4 : ARRn);

VAR
  locus1, locus2 : INTEGER;
  aux3, aux4     : INTEGER;

BEGIN          (* MUTATION2 *)

  RANDOMIZE;

  REPEAT
    locus1 := random(n-1);
    locus1 := locus1 + 2;
    locus2 := random(n-1);
    locus2 := locus2 + 2;
  UNTIL (locus1 < locus2);

  aux3 := s3[locus1];

```

```

s3[locus1] := s3[locus2];
s3[locus2] := aux3;

aux4 := s4[locus1];
s4[locus1] := s4[locus2];
s4[locus2] := aux4;
END;          (* MUTATION2 *)

```

```

PROCEDURE MUTATION3(
  VAR s3, s4 : ARR);

```

```

  VAR
    cut, i : INTEGER;
    AUXs3, AUXs4 : ARR;

  BEGIN          (* MUTATION3 *)

```

```

    RANDOMIZE;

```

```

    cut := random(n-2);
    cut := cut + 2;

```

```

    AUXs3[1] := s3[1]; AUXs4[1] := s4[1];
    FOR i := 2 TO (n-cut+1) DO

```

```

      BEGIN
        AUXs3[i] := s3[i+cut-1];
        AUXs4[i] := s4[i+cut-1];

```

```

      END;
      FOR i := (n-cut+2) TO n DO

```

```

        BEGIN
          AUXs3[i] := s3[i+cut-n];
          AUXs4[i] := s4[i+cut-n];
        END;

```

```

    s3 := AUXs3 ;
    s4 := AUXs4 ;

```

```

  END;          (* MUTATION3 *)

```

```

PROCEDURE SORTsequences(
  VAR D1, D2, D3, D4 : INTEGER;
  VAR s1, s2 : ARR;
  VAR s3, s4 : ARR);

```

```

  VAR
    AuxD : INTEGER;
    AuxS : ARR;

```

```

  BEGIN          (* SORTsequences *)

```

```

    IF D1 >= D2 THEN
      BEGIN
        AuxD := D2 ; AuxS := s2 ;
        D2 := D1 ; s2 := s1 ;
        D1 := AuxD ; s1 := AuxS ;
      END;

```

```

IF D2 >= D3 THEN
  BEGIN
    AuxD := D3 ; AuxS := s3 ;
    D3 := D2 ; s3 := s2 ;
    D2 := AuxD ; s2 := AuxS ;
  END;

IF D1 >= D2 THEN
  BEGIN
    AuxD := D2 ; AuxS := s2 ;
    D2 := D1 ; s2 := s1 ;
    D1 := AuxD ; s1 := AuxS ;
  END;

IF D3 >= D4 THEN
  BEGIN
    AuxD := D4 ; AuxS := s4 ;
    D4 := D3 ; s4 := s3 ;
    D3 := AuxD ; s3 := AuxS ;
  END;

IF D2 >= D3 THEN
  BEGIN
    AuxD := D3 ; AuxS := s3 ;
    D3 := D2 ; s3 := s2 ;
    D2 := AuxD ; s2 := AuxS ;
  END;
IF D1 >= D2 THEN
  BEGIN
    AuxD := D2 ; AuxS := s2 ;
    D2 := D1 ; s2 := s1 ;
    D1 := AuxD ; s1 := AuxS ;
  END;

END;          (* SORTsequences *)

PROCEDURE SAforHybrid(
  VAR P      : ARRNM;
  VAR CSeq   : ARRNM;
  VAR DCSeq  : INTEGER;
  VAR BS     : ARRNM;
  VAR BD     : INTEGER;
  VAR saBS   : ARRNM;
  VAR saBD   : INTEGER;
  VAR NbBestSeq1 : INTEGER;
  VAR Nbl    : INTEGER);

  VAR
    S, NBCSeq, NewCSeq   : ARRNM;
    i, j                 : BYTE;
    K                    : INTEGER;
    DNBCSeq, DNewCSeq   : INTEGER;
    NbSeq1               : INTEGER;

  BEGIN          (* SAforHybrid *)

    K := NbHybridSAsequences( n, Percent );

```

```

B := (T1-TK) / ((K-1)*T1*TK);

saBS := CSeq; saBD := DCSeq; Temp := T1;

NbSeq1 := 0 ; NbBestSeq1 := 0 ;

WHILE NbSeq1 < K DO

BEGIN
S := CSeq;
GetRandomSAShiftNeighbour(S, NBCSeq);
NbSeq1 := NbSeq1 + 1;

DCSeq := MAKESPAN(m, n, CSeq, P);
DNBCSeq := MAKESPAN(m, n, NBCSeq, P);

GetSACurrentSequence(CSeq,NBCSeq,NewCSeq,Temp,DCSeq,DNBCSeq,DNewCSeq);
CSeq := NewCSeq ;

IF DNewCSeq < saBD THEN
BEGIN
saBS := NewCSeq;
saBD := DNewCSeq;
END;

IF saBD < BD THEN
BEGIN
BS := saBS;
BD := saBD;
NbI := NbI + 1 ;
NbBestSeq1 := NbSeq1;
END;

Temp := NextTemp(Temp, B);

END;
END; (* SAforHybrid *)

```

SOLUÇÃO INICIAL DO MÉTODO FSHOPH

```

PROCEDURE FITSP(
S : INTEGER;
VAR N : INTEGER;
VAR TWEIGHT : INTEGER;
VAR ROUTE : ARRNN;
VAR W : ARRNN);

VAR
END1, END2, FARTHEST, I,
INDEX, J, NEXTINDEX : INTEGER;
INSCOST, MAXDIST, NEWCOST : INTEGER;
CYCLE, DIST : ARRNN;

```



```

BEGIN                                     (* FITSP *)

FOR I := 1 TO N DO CYCLE[I] := 0 ;
CYCLE[S] := S ;
W[S,S] := 0 ;
FOR I := 1 TO N DO DIST[I] := W[S,I] ;
TWEIGHT := 0 ;
FOR I := 1 TO (N-1) DO
  BEGIN
    MAXDIST := -INF ;
    FOR J := 1 TO N DO
      IF CYCLE[J] = 0 THEN
        IF DIST[J] > MAXDIST THEN
          BEGIN
            MAXDIST := DIST[J] ;
            FARTHEST := J
          END ;
        INSCOST := INF ;
        INDEX := S ;
        FOR J := 1 TO I DO
          BEGIN
            NEXTINDEX := CYCLE[INDEX] ;
            NEWCOST := W[INDEX,FARTHEST] +
              W[FARTHEST,NEXTINDEX] -
              W[INDEX,NEXTINDEX] ;
            IF NEWCOST < INSCOST THEN
              BEGIN
                INSCOST := NEWCOST ;
                END1 := INDEX ;
                END2 := NEXTINDEX
              END ;
            INDEX := NEXTINDEX
          END ;
          CYCLE[FARTHEST] := END2 ;
          CYCLE[END1] := FARTHEST ;
          TWEIGHT := TWEIGHT + INSCOST ;
          FOR J := 1 TO N DO
            IF CYCLE[J] = 0 THEN
              IF W[FARTHEST,J] < DIST[J] THEN
                DIST[J] := W[FARTHEST,J]
            END ;
          INDEX := S ;
        FOR I := 1 TO N DO
          BEGIN
            ROUTE[I] := INDEX ;
            INDEX := CYCLE[INDEX]
          END
        END ;
  END ;
END;                                     (* FITSP *)

```

ALGORITMO NEH

```

PROCEDURE NEH(
  VAR m,n      : INTEGER;
  VAR P        : ARRNM;
  VAR SEQUENCE : ARRNM;
  VAR D        : INTEGER);

VAR
  i      : INTEGER;
  LPTSeq : ARRNM;

PROCEDURE LPTJobSort(VAR LPTSeq : ARRNM);

TYPE
  TPJob = RECORD
    index, TP : INTEGER;
  END;
  TPList = ARRAY[1..101] OF TPJob;

VAR
  i, u, v : INTEGER;
  TPL     : TPList;

PROCEDURE QUICKSORT(start, finish : INTEGER);

VAR
  before, after, midPoint : INTEGER;
  aux                  : TPJob;

BEGIN
  (* QUICKSORT *)
  bbefore := start;
  after := finish;
  midPoint := TPL[(start+finish) div 2].TP;
  REPEAT
    WHILE TPL[before].TP > midPoint DO
      before := before+1;
    WHILE midPoint > TPL[after].TP DO
      after := after-1;
    IF before <= after THEN
      BEGIN
        aux := TPL[before];
        TPL[before] := TPL[after];
        TPL[after] := aux;
        before := before+1;
        after := after-1;
      END
    UNTIL bbefore > after;

    IF start < after THEN
      QUICKSORT(start, after);
    IF before < finish THEN
      QUICKSORT(bbefore, finish);
  END;
  (* QUICKSORT *)

```

```

BEGIN          (* LPTJobSort *)

  FOR v := 1 TO n DO
    BEGIN
      TPL[v].index := v; TPL[v].TP := 0;
      FOR u := 1 TO m DO
        TPL[v].TP := TPL[v].TP + P[u,v];
      END;

    QUICKSORT(2,n);
    FOR i := 1 TO n DO LPTSeq[i] := TPL[i].index;
  END;          (* LPTJobSort *)

PROCEDURE NEHJobInsertionSequence(
  VAR N      : INTEGER;
  VAR S, SEQUENCE : ARR);

  VAR I, J, K, Z, StartN : INTEGER;
  StartD, InsD, NewD, NbJob : INTEGER;
  INDEX, NEXTINDEX, InsJob : INTEGER;
  END1, END2 : INTEGER;
  StartSeq, PartialSeq : ARR;
  PartialCYCLE, CYCLE : ARR;

  BEGIN          (* NEHJobInsertionSequence *)
    StartN := 3;
    StartSeq[1] := S[1];
    StartSeq[2] := S[2]; StartSeq[3] := S[3];
    StartD := MAKESPAN(m, StartN, StartSeq, P);

    StartSeq[2] := S[3]; StartSeq[3] := S[2];
    IF StartD < MAKESPAN(m, StartN, StartSeq, P) THEN

      BEGIN
        StartSeq[2] := S[2]; StartSeq[3] := S[3];
      END;

    FOR I := 1 TO N DO CYCLE[I] := 0;
    CYCLE[1] := StartSeq[2];
    CYCLE[StartSeq[2]] := StartSeq[3];
    CYCLE[StartSeq[3]] := 1;

    FOR K := 3 TO (N-1) DO
      BEGIN
        InsJob := S[K+1];
        NbJob := K+1;
        InsD := INF;
        INDEX := 1;
        FOR I := 1 TO N DO
          PartialCYCLE[I] := CYCLE[I];

        FOR J := 1 TO K DO
          BEGIN
            NEXTINDEX := PartialCYCLE[INDEX];
            PartialCYCLE[INDEX] := InsJob;
            PartialCYCLE[InsJob] := NEXTINDEX;
            Z := 1;

```

```

FOR I := 1 TO NbJob DO
  BEGIN
    PartialSeq[I] := Z;
    Z := PartialCYCLE[Z];
    END;
NewD := MAKESPAN(m,NbJob,PartialSeq,P);

IF NewD < InsD THEN
  BEGIN
    InsD := NewD;
    END1 := INDEX;
    END2 := NEXTINDEX;
    END;
INDEX := NEXTINDEX;
FOR I := 1 TO N DO
  PartialCYCLE[I] := CYCLE[I];
  END;
CYCLE[InsJob] := END2;
CYCLE[END1] := InsJob;
  END;

INDEX := 1;
FOR I := 1 TO N DO
  BEGIN
    SEQUENCE[I] := INDEX;
    INDEX := CYCLE[INDEX];
  END;
END;          (* NEHJobInsertionSequence *)

BEGIN          (* NEH *)

  FOR i := 1 TO n DO SEQUENCE[i] := 1;
  D := 0;

  LPTJobSort(LPTSeq);

  NEHJobInsertionSequence(n,LPTSeq,SEQUENCE);

  D := MAKESPAN(m,n,SEQUENCE,P);

END;          (* NEH *)

```

ALGORITMO GENÉTICO (OPERADOR DE 1 CORTE)

```

PROCEDURE onecutGA(
  VAR P          : ARRMN;
  VAR CSeq      : StartSEQS;
  VAR DCSeq1, DCSeq2 : INTEGER;
  VAR BS       : ARRN;
  VAR BD       : INTEGER;
  VAR NbSeq, NbBestSeq : LONGINT;
  VAR SR       : REAL;
  VAR Nbl      : INTEGER;
  VAR PR       : REAL);

```

```

VAR
  K                : LONGINT;
  NbMut1, NbMut2, NbMut3  : LONGINT;
  Mut1, Mut2, Mut3      : INTEGER;
  seq1, seq2, seq3, seq4  : ARR;
  Dseq1, Dseq2, Dseq3, Dseq4 : INTEGER;

  i, SameJobs        : INTEGER;
  SameParents        : BOOLEAN;

BEGIN
  (* onecutGA *)

  seq1 := CSeq[1]; seq2 := CSeq[2];
  NbSeq := 0 ; NbBestSeq := 0 ;
  SR := 0.0 ; Nbl := 0 ; PR := 0.0 ;
  K := 0 ;
  NbMut1 := 0 ; NbMut2 := 0 ; NbMut3 := 0 ;

  IF DCSeq1 <= DCSeq2 THEN
    BEGIN
      BS := CSeq[1] ;
      BD := DCSeq1 ;
    END
  ELSE
    BEGIN
      BS := CSeq[2] ;
      BD := DCSeq2 ;
    END;

  NbSequencesGAandModSAandHybrid( m, n, K );
  Mut1 := ROUND(0.01*K) ;
  Mut2 := ROUND(0.027*K) ;
  Mut3 := ROUND(0.10*K) ;

  SameParents := FALSE ;

REPEAT

  IF (NbSeq - NbMut1) <= Mut1 THEN
    BEGIN
      onecutOPERATOR( n, seq1, seq2, seq3, seq4 );
      NbSeq := NbSeq + 2 ;
      Dseq1 := MAKESPAN( m, n, seq1, P );
      Dseq2 := MAKESPAN( m, n, seq2, P );
      Dseq3 := MAKESPAN( m, n, seq3, P );
      Dseq4 := MAKESPAN( m, n, seq4, P );

      IF Dseq3 <= Dseq4 THEN
        BEGIN
          IF Dseq3 < BD THEN
            BEGIN
              BS := seq3 ;
              BD := Dseq3 ;
              Nbl := Nbl + 1 ;
              NbBestSeq := NbSeq ;
              NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
            END;
          END
        END
      END
    END
  END

```

```

ELSE
  IF Dseq4 < BD THEN
    BEGIN
      BS := seq4 ;
      BD := Dseq4 ;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
      NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
    END;

  SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

  IF Dseq1 < BD THEN
    BEGIN
      BS := seq1;
      BD := Dseq1;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
    END;

  SameJobs := 0 ;

  FOR i := 1 TO n DO
    IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
    IF SameJobs = n THEN SameParents := TRUE ;
  END;

  IF (NbSeq - NbMut1) > Mut1 THEN
    BEGIN
      onecutOPERATOR( n, seq1, seq2, seq3, seq4 );
      NbSeq := NbSeq + 2 ;

      Dseq3 := MAKESPAN( m, n, seq3, P );
      Dseq4 := MAKESPAN( m, n, seq4, P );

      IF Dseq3 < BD THEN
        BEGIN
          BS := seq3;
          BD := Dseq3;
          NbI := NbI + 1 ;
          NbBestSeq := NbSeq ;
        END;
      IF Dseq4 < BD THEN
        BEGIN
          BS := seq4;
          BD := Dseq4;
          NbI := NbI + 1 ;
          NbBestSeq := NbSeq ;
        END;

      MUTATION1( seq3, seq4 );
      NbSeq := NbSeq + 2 ;
      NbMut1 := NbSeq ;

      Dseq1 := MAKESPAN( m, n, seq1, P );
      Dseq2 := MAKESPAN( m, n, seq2, P );
      Dseq3 := MAKESPAN( m, n, seq3, P );

```

```

Dseq4 := MAKESPAN( m, n, seq4, P );

IF Dseq3 <= Dseq4 THEN
  BEGIN
    IF Dseq3 < BD THEN
      BEGIN
        BS := seq3 ;
        BD := Dseq3 ;
        Nbl := Nbl + 1 ;
        NbBestSeq := NbSeq ;
        NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
      END;
    END
  ELSE
    IF Dseq4 < BD THEN
      BEGIN
        BS := seq4 ;
        BD := Dseq4 ;
        Nbl := Nbl + 1 ;
        NbBestSeq := NbSeq ;
        NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
      END;

SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

IF Dseq1 < BD THEN
  BEGIN
    BS := seq1;
    BD := Dseq1;
    Nbl := Nbl + 1 ;
    NbBestSeq := NbSeq ;
  END;

SameJobs := 0 ;

FOR i := 1 TO n DO
  IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
  IF SameJobs = n THEN SameParcnts := TRUE ;

END;

IF (NbSeq - NbMut2) > Mut2 THEN
  BEGIN
    onccutOPERATOR( n, seq1, seq2, seq3, seq4 );
    NbSeq := NbSeq + 2 ;

    Dseq3 := MAKESPAN( m, n, seq3, P );
    Dseq4 := MAKESPAN( m, n, seq4, P );

    IF Dseq3 < BD THEN
      BEGIN
        BS := seq3;
        BD := Dseq3;
        Nbl := Nbl + 1 ;
        NbBestSeq := NbSeq ;
      END;

    IF Dseq4 < BD THEN

```

```

BEGIN
  BS := seq4;
  BD := Dseq4;
  NbI := NbI + 1 ;
  NbBestSeq := NbSeq ;
END;

MUTATION2( seq3, seq4 );
NbSeq := NbSeq + 2 ;
NbMut2 := NbSeq ;
NbMut1 := NbSeq ;

Dseq1 := MAKESPAN( m, n, seq1, P );
Dseq2 := MAKESPAN( m, n, seq2, P );
Dseq3 := MAKESPAN( m, n, seq3, P );
Dseq4 := MAKESPAN( m, n, seq4, P );

IF Dseq3 <= Dseq4 THEN
  BEGIN
    IF Dseq3 < BD THEN
      BEGIN
        BS := seq3 ;
        BD := Dseq3 ;
        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
        NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
      END;
    END
  ELSE
    IF Dseq4 < BD THEN
      BEGIN
        BS := seq4 ;
        BD := Dseq4 ;
        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
        NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
      END;
    END;

  SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

  IF Dseq1 < BD THEN
    BEGIN
      BS := seq1;
      BD := Dseq1;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
    END;

  SameJobs := 0 ;

  FOR i := 1 TO n DO
    IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
    IF SameJobs = n THEN SameParents := TRUE ;
  END;

  IF (NbSeq - NbMut3) > Mut3 THEN
    BEGIN

```



```

onecutOPERATOR( n, seq1, seq2, seq3, seq4 );
NbSeq := NbSeq + 2 ;

Dseq3 := MAKESPAN( m, n, seq3, P );
Dseq4 := MAKESPAN( m, n, seq4, P );

IF Dseq3 < BD THEN
  BEGIN
    BS := seq3;
    BD := Dseq3;
    NbI := NbI + 1 ;
    NbBestSeq := NbSeq ;
  END;

IF Dseq4 < BD THEN
  BEGIN
    BS := seq4;
    BD := Dseq4;
    NbI := NbI + 1 ;
    NbBestSeq := NbSeq ;
  END;

MUTATION3( seq3, seq4 );
NbSeq := NbSeq + 2 ;
NbMut3 := NbSeq ;
NbMut2 := NbSeq ;
NbMut1 := NbSeq ;

Dseq1 := MAKESPAN( m, n, seq1, P );
Dseq2 := MAKESPAN( m, n, seq2, P );
Dseq3 := MAKESPAN( m, n, seq3, P );
Dseq4 := MAKESPAN( m, n, seq4, P );

IF Dseq3 <= Dseq4 THEN
  BEGIN
    IF Dseq3 < BD THEN
      BEGIN
        BS := seq3 ;
        BD := Dseq3 ;
        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
      END;
    END
  ELSE
    IF Dseq4 < BD THEN
      BEGIN
        BS := seq4 ;
        BD := Dseq4 ;
        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
      END;
    END;

SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

IF Dseq1 < BD THEN
  BEGIN
    BS := seq1;
    BD := Dseq1;
  
```

```

        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
    END;

    SameJobs := 0 ;

    FOR i := 1 TO n DO
        IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
        IF SameJobs = n THEN SameParents := TRUE ;
    END;

UNTIL ( NbSeq >= K ) OR ( SameParents );

IF NbSeq >= K THEN
    BEGIN
        SR := 1.0 ;
        PR := NbI / NbSeq ;
    END
ELSE
    BEGIN
        SR := NbSeq / K ;
        PR := NbI / K ;
    END;
END;
(* onecutGA *)

```

ALGORITMO GENÉTICO (OPERADOR PMX)

```

PROCEDURE pmxGA(
    VAR P          : ARRNM;
    VAR CSeq       : StartSEQS;
    VAR DCSeq1, DCSeq2 : INTEGER;
    VAR BS         : ARRNM;
    VAR BD         : INTEGER;
    VAR NbSeq, NbBestSeq : LONGINT;
    VAR SR         : REAL;
    VAR NbI        : INTEGER;
    VAR PR         : REAL);
VAR
    K          : LONGINT;
    NbMut1, NbMut2, NbMut3 : LONGINT;
    Mut1, Mut2, Mut3      : INTEGER;
    seq1, seq2, seq3, seq4 : ARRNM;
    Dseq1, Dseq2, Dseq3, Dseq4 : INTEGER;
    i, SameJobs          : INTEGER;
    SameParents          : BOOLEAN;

BEGIN
    (* pmxGA *)

    seq1 := CSeq[1]; seq2 := CSeq[2];
    NbSeq := 0 ; NbBestSeq := 0 ;
    SR := 0.0 ; NbI := 0 ; PR := 0.0 ;
    K := 0 ;
    NbMut1 := 0 ; NbMut2 := 0 ; NbMut3 := 0 ;

```

```

IF DCSeq1 <= DCSeq2 THEN
  BEGIN
    BS := CSeq[1];
    BD := DCSeq1;
  END
ELSE
  BEGIN
    BS := CSeq[2];
    BD := DCSeq2;
  END;

NbSequencesGAandModSAandHybrid( m, n, K );
Mut1 := ROUND(0.01*K);
Mut2 := ROUND(0.027*K);
Mut3 := ROUND(0.10*K);

SameParents := FALSE;

REPEAT

  IF (NbSeq - NbMut1) <= Mut1 THEN
    BEGIN
      pmxOPERATOR( n, seq1, seq2, seq3, seq4 );
      NbSeq := NbSeq + 2;

      Dseq1 := MAKESPAN( m, n, seq1, P );
      Dseq2 := MAKESPAN( m, n, seq2, P );
      Dseq3 := MAKESPAN( m, n, seq3, P );
      Dseq4 := MAKESPAN( m, n, seq4, P );

      IF Dseq3 <= Dseq4 THEN
        BEGIN
          IF Dseq3 < BD THEN
            BEGIN
              BS := seq3;
              BD := Dseq3;
              NbI := NbI + 1;
              NbBestSeq := NbSeq;
              NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
            END;
          END
        ELSE
          IF Dseq4 < BD THEN
            BEGIN
              BS := seq4;
              BD := Dseq4;
              NbI := NbI + 1;
              NbBestSeq := NbSeq;
              NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
            END;
          END
        END;

      SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

      IF Dseq1 < BD THEN
        BEGIN
          BS := seq1;
          BD := Dseq1;
          NbI := NbI + 1;

```

```

    NbBestSeq := NbSeq ;
  END;

  SameJobs := 0 ;

  FOR i := 1 TO n DO
    IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
    IF SameJobs = n THEN SameParents := TRUE ;
  END;

  IF (NbSeq - NbMut1) > Mut1 THEN
  BEGIN
    pmxOPERATOR( n, seq1, seq2, seq3, seq4 );
    NbSeq := NbSeq + 2 ;

    Dseq3 := MAKESPAN( m, n, seq3, P );
    Dscq4 := MAKESPAN( m, n, scq4, P );

    IF Dseq3 < BD THEN
    BEGIN
      BS := seq3;
      BD := Dscq3;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
    END;

    IF Dseq4 < BD THEN
    BEGIN
      BS := seq4;
      BD := Dseq4;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
    END;

    MUTATION1( seq3, seq4 );
    NbSeq := NbSeq + 2 ;
    NbMut1 := NbSeq ;

    Dseq1 := MAKESPAN( m, n, seq1, P );
    Dseq2 := MAKESPAN( m, n, seq2, P );
    Dscq3 := MAKESPAN( m, n, scq3, P );
    Dseq4 := MAKESPAN( m, n, seq4, P );

    IF Dseq3 <= Dseq4 THEN
    BEGIN
      IF Dseq3 < BD THEN
      BEGIN
        BS := seq3 ;
        BD := Dseq3 ;
        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
        NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
      END;
    END
  ELSE
    IF Dseq4 < BD THEN

```

```

BEGIN
  BS := seq4 ;
  BD := Dseq4 ;
  Nbl := Nbl + 1 ;
  NbBestSeq := NbSeq ;
  NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
END;

SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

IF Dseq1 < BD THEN
  BEGIN
    BS := seq1;
    BD := Dseq1;
    Nbl := Nbl + 1 ;
    NbBestSeq := NbSeq ;
  END;

  SameJobs := 0 ;

  FOR i := 1 TO n DO
    IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
    IF SameJobs = n THEN SameParents := TRUE ;
  END;

IF (NbSeq - NbMut2) > Mut2 THEN
  BEGIN
    pmxOPERATOR( n, seq1, seq2, seq3, seq4 );
    NbSeq := NbSeq + 2 ;

    Dseq3 := MAKESPAN( m, n, seq3, P );
    Dseq4 := MAKESPAN( m, n, seq4, P );

    IF Dseq3 < BD THEN
      BEGIN
        BS := seq3;
        BD := Dseq3;
        Nbl := Nbl + 1 ;
        NbBestSeq := NbSeq ;
      END;
    IF Dseq4 < BD THEN
      BEGIN
        BS := seq4;
        BD := Dseq4;
        Nbl := Nbl + 1 ;
        NbBestSeq := NbSeq ;
      END;

    MUTATION2( seq3, seq4 );
    NbSeq := NbSeq + 2 ;
    NbMut2 := NbSeq ;
    NbMut1 := NbSeq ;

    Dseq1 := MAKESPAN( m, n, seq1, P );
    Dseq2 := MAKESPAN( m, n, seq2, P );
    Dseq3 := MAKESPAN( m, n, seq3, P );
    Dseq4 := MAKESPAN( m, n, seq4, P );

```

```

IF Dseq3 <= Dseq4 THEN
  BEGIN
  IF Dseq3 < BD THEN
    BEGIN
      BS := seq3 ;
      BD := Dseq3 ;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
      NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
    END;
  END
  ELSE
  IF Dseq4 < BD THEN
    BEGIN
      BS := seq4 ;
      BD := Dseq4 ;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
      NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
    END;

  SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

  IF Dseq1 < BD THEN
    BEGIN
      BS := seq1;
      BD := Dseq1;
      NbI := NbI + 1 ;
      NbBestSeq := NbSeq ;
    END;

  SameJobs := 0 ;

  FOR i := 1 TO n DO
    IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
    IF SameJobs = n THEN SameParents := TRUE ;

  END;

  IF (NbSeq - NbMut3) > Mut3 THEN
  BEGIN
    pmxOPERATOR( n, seq1, seq2, seq3, seq4 );
    NbSeq := NbSeq + 2 ;

    Dseq3 := MAKESPAN( m, n, seq3, P );
    Dseq4 := MAKESPAN( m, n, seq4, P );

    IF Dseq3 < BD THEN
      BEGIN
        BS := seq3;
        BD := Dseq3;
        NbI := NbI + 1 ;
        NbBestSeq := NbSeq ;
      END;

    IF Dseq4 < BD THEN
      BEGIN
        BS := seq4;

```

```

    BD := Dseq4;
    NbI := NbI + 1 ;
    NbBestSeq := NbSeq ;
END;

MUTATION3( seq3, seq4 );
NbSeq := NbSeq + 2 ;
NbMut3 := NbSeq ;
NbMut2 := NbSeq ;
NbMut1 := NbSeq ;

Dseq1 := MAKESPAN( m, n, seq1, P );
Dseq2 := MAKESPAN( m, n, seq2, P );
Dseq3 := MAKESPAN( m, n, seq3, P );
Dseq4 := MAKESPAN( m, n, seq4, P );

IF Dseq3 <= Dseq4 THEN
BEGIN
  IF Dseq3 < BD THEN
  BEGIN
    BS := seq3 ;
    BD := Dseq3 ;
    NbI := NbI + 1 ;
    NbBestSeq := NbSeq ;
  END;
END
ELSE
  IF Dseq4 < BD THEN
  BEGIN
    BS := seq4 ;
    BD := Dseq4 ;
    NbI := NbI + 1 ;
    NbBestSeq := NbSeq ;
  END;

SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

IF Dseq1 < BD THEN
  BEGIN
    BS := seq1;
    BD := Dseq1;
    NbI := NbI + 1 ;
    NbBestSeq := NbSeq ;
  END;

SameJobs := 0 ;

FOR i := 1 TO n DO
  IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
  IF SameJobs = n THEN SameParents := TRUE ;

END;

UNTIL ( NbSeq >= K ) OR ( SameParents ) ;

IF NbSeq >= K THEN
  BEGIN
    SR := 1.0 ;

```

```

    PR := NbI / NbSeq ;
  END
ELSE
  BEGIN
    SR := NbSeq / K ;
    PR := NbI / K ;
  END;
END;
(* pmxGA *)

```

ALGORITMO modSAfshopH

```

PROCEDURE modSAfshopH (
  VAR P      : ARRNM;
  VAR CSeq   : ARRNM;
  VAR DCSeq  : INTEGER;
  VAR BS     : ARRNM;
  VAR BD     : INTEGER;
  VAR NbSeq  : LONGINT;
  VAR NbBestSeq : LONGINT;
  VAR SR     : REAL;
  VAR NbI    : INTEGER;
  VAR PR     : REAL);

  VAR
    S, NBCSeq, NewCSeq   : ARRNM;
    i, j                 : BYTE;
    K                    : LONGINT;
    DNBCSeq, DNewCSeq   : INTEGER;

  BEGIN
    (* modSAfshopH *)

    NbSequencesGAandModSAandHybrid( m, n, K );

    B := (T1-TK) / ((K-1)*T1*TK);

    BS := CSeq; BD := DCSeq; Temp := T1;
    NbSeq := 0; NbBestSeq := 0;
    SR := 0.0 ; NbI := 0 ; PR := 0.0 ;

    WHILE NbSeq < K DO

      BEGIN
        S := CSeq;
        GetRandomSAShiftNeighbour(S, NBCSeq);
        NbSeq := NbSeq + 1;

        DCSeq := MAKESPAN(m, n, CSeq, P);
        DNBCSeq := MAKESPAN(m, n, NBCSeq, P);

        GetSACurrentSequence(CSeq, NBCSeq, NewCSeq, Temp, DCSeq, DNBCSeq, DNewCSeq);
        CSeq := NewCSeq ;

        IF DNewCSeq < BD THEN

```



```

BEGIN
  BS := NewCSeq;
  BD := DNewCSeq;
  NbI := NbI + 1 ;
  NbBestSeq := NbSeq ;
  END;

  Temp := NextTemp(Temp, B);

  END;
  SR := 1.0 ;
  PR := NbI / NbSeq ;
  END;                                     (* modSAfshopH *)

```

ALGORITMO HÍBRIDO HBGASA (OPERADOR DE 1 CORTE)

```

PROCEDURE Hybrid1XGAandModSA(
  VAR P      : ARRNM;
  VAR CSeq   : StartSEQS;
  VAR DCSeq1, DCSeq2 : INTEGER;
  VAR BS     : ARRNM;
  VAR BD     : INTEGER;
  VAR NbSeq  : LONGINT;
  VAR NbBestSeq : LONGINT;
  VAR SR     : REAL;
  VAR NbI    : INTEGER;
  VAR PR     : REAL);

  VAR
    K      : LONGINT;
    seq1, seq2, seq3, seq4 : ARRNM;
    Dseq1, Dseq2, Dseq3, Dseq4 : INTEGER;
    NewSeq1, NewSeq2 : ARRNM;
    DNewSeq1, DNewSeq2 : INTEGER;
    IncNbSeq, IncNbBestSeq : INTEGER;

    i, SameJobs : INTEGER;
    SameParents : BOOLEAN;

  BEGIN                                     (* Hybrid1XGAandModSA *)

    seq1 := CSeq[1]; seq2 := CSeq[2];
    NbSeq := 0 ; NbBestSeq := 0 ;
    SR := 0.0 ; NbI := 0 ; PR := 0.0 ;

    IF DCSeq1 <= DCSeq2 THEN
      BEGIN
        BS := CSeq[1] ;
        BD := DCSeq1 ;
      END
    ELSE
      BEGIN
        BS := CSeq[2] ;
        BD := DCSeq2 ;
      END;
    END;

```

```

NbSequencesGAandModSAandHybrid( m, n, K );
IncNbSeq := NbHybridSAsequences( n, Percent );

SameParents := FALSE ;

REPEAT
  onecutOPERATOR( n, seq1, seq2, seq3, seq4 );
  NbSeq := NbSeq + 2 ;

  Dseq1 := MAKESPAN( m, n, seq1, P );
  Dseq2 := MAKESPAN( m, n, seq2, P );
  Dseq3 := MAKESPAN( m, n, seq3, P );
  Dseq4 := MAKESPAN( m, n, seq4, P );

  SORTscqucnccs(Dscq1,Dscq2,Dscq3,Dscq4,scq1,seq2,seq3,scq4 );

  IF Dseq1 < BD THEN
    BEGIN
      BS := seq1 ;
      BD := Dscq1 ;
      Nbl := Nbl + 1 ;
      NbBestSeq := NbSeq ;
    END;

  SAforHybrid( P, seq1, Dseq1, BS, BD, NewSeq1, DNewSeq1, IncNbBestSeq, Nbl);
  IF IncNbBestSeq > 0 THEN NbBestSeq := NbSeq + IncNbBestSeq ;
  NbSeq := NbSeq + IncNbSeq ;

  SAforHybrid( P, seq2, Dseq2, BS, BD, NewSeq2, DNewSeq2, IncNbBestSeq, Nbl);
  IF IncNbBestSeq > 0 THEN NbBestSeq := NbSeq + IncNbBestSeq ;
  NbSeq := NbSeq + IncNbSeq ;

  seq1 := NewSeq1 ;
  seq2 := NewSeq2 ;

  SameJobs := 0 ;

  FOR i := 1 TO n DO
    IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
    IF SameJobs = n THEN SameParents := TRUE ;

  UNTIL ( NbSeq >= K ) OR ( SameParents ) ;

  IF NbSeq >= K THEN
    BEGIN
      SR := 1.0 ;
      PR := Nbl / NbSeq ;
    END
  ELSE
    BEGIN
      SR := NbSeq / K ;
      PR := Nbl / K ;
    END;
  END;
END;
(* Hybrid1XGAandModSA *)

```

ALGORITMO HÍBRIDO HBGASA (OPERADOR PMX)

```

PROCEDURE HybridPmxGAandModSA(
  VAR P      : ARRNM;
  VAR CSeq   : StartSEQS;
  VAR DCSeq1, DCSeq2 : INTEGER;
  VAR BS     : ARRNM;
  VAR BD     : INTEGER;
  VAR NbSeq  : LONGINT;
  VAR NbBestSeq : LONGINT;
  VAR SR     : REAL;
  VAR NbI    : INTEGER;
  VAR PR     : REAL);

VAR
  K      : LONGINT;
  seq1, seq2, seq3, seq4 : ARRNM;
  Dseq1, Dseq2, Dseq3, Dseq4 : INTEGER;
  NewSeq1, NewSeq2 : ARRNM;
  DNewSeq1, DNewSeq2 : INTEGER;
  IncNbSeq, IncNbBestSeq : INTEGER;
  i, SameJobs : INTEGER;
  SameParents : BOOLEAN;

BEGIN
  (* HybridPmxGAandModSA *)

  seq1 := CSeq[1]; seq2 := CSeq[2];
  NbSeq := 0; NbBestSeq := 0;
  SR := 0.0; NbI := 0; PR := 0.0;

  IF DCSeq1 <= DCSeq2 THEN
    BEGIN
      BS := CSeq[1];
      BD := DCSeq1;
    END
  ELSE
    BEGIN
      BS := CSeq[2];
      BD := DCSeq2;
    END;

  NbSequencesGAandModSAandHybrid( m, n, K );
  IncNbSeq := NbHybridSAsequences( n, Percent );

  SameParents := FALSE;

  REPEAT

    pmxOPERATOR( n, seq1, seq2, seq3, seq4 );
    NbSeq := NbSeq + 2;

    Dseq1 := MAKESPAN( m, n, seq1, P );
    Dseq2 := MAKESPAN( m, n, seq2, P );
    Dseq3 := MAKESPAN( m, n, seq3, P );
    Dseq4 := MAKESPAN( m, n, seq4, P );

    SORTsequences(Dseq1,Dseq2,Dseq3,Dseq4,seq1,seq2,seq3,seq4 );

```

```

IF Dseq1 < BD THEN
  BEGIN
    BS := seq1 ;
    BD := Dseq1 ;
    Nbl := Nbl + 1 ;
    NbBestSeq := NbSeq ;
  END;

SAforHybrid( P, seq1, Dseq1, BS, BD, NewSeq1, DNewSeq1, IncNbBestSeq, Nbl);
IF IncNbBestSeq > 0 THEN NbBestSeq := NbSeq + IncNbBestSeq ;
NbSeq := NbSeq + IncNbSeq ;

SAforHybrid( P, seq2, Dscq2, BS, BD, NewScq2, DNewScq2, IncNbBcstSeq, Nbl);
IF IncNbBestSeq > 0 THEN NbBestSeq := NbSeq + IncNbBestSeq ;
NbSeq := NbSeq + IncNbSeq ;

seq1 := NewSeq1 ;
scq2 := NcwScq2 ;

SameJobs := 0 ;

FOR i := 1 TO n DO
  IF seq1[i] = seq2[i] THEN SameJobs := SameJobs + 1 ;
  IF SameJobs = n THEN SameParents := TRUE ;

UNTIL ( NbSeq >= K ) OR ( SameParents ) ;

IF NbSeq >= K THEN
  BEGIN
    SR := 1.0 ;
    PR := Nbl / NbSeq ;
  END
ELSE
  BEGIN
    SR := NbSeq / K ;
    PR := Nbl / K ;
  END;

END;

(* HybridPmxGAandModSA *)

```

REFERÊNCIAS BIBLIOGRÁFICAS

- BURBIDGE, J.L. (1986). *Planejamento e Controle da Produção*. São Paulo, Atlas.
- CAMPBELL, H.G.; DUDEK, R.A.; SMITH, M.L. (1970). A Heuristic Algorithm for the n-job, m-machine Sequencing Problem. *Management Science*, v.16/B, p.630-637.
- DANNENBRING, D.G. (1977). An Evaluation of Flow-Shop Sequencing Heuristics. *Management Science*, v.23, p.1174-1182.
- DÍAZ, B.A. (1996). An SA/TS Mixture Algorithm for the Scheduling Tardiness Problem. *European Journal of Operational Research*, v. 88, p. 516-524.
- FERNANDES, F.C.F. (1991). *Concepção de um Sistema de Controle da Produção Para Manufatura Celular*. São Carlos. 239p. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.
- GAREY, M.R.; JOHNSON, D.S.; SETHI, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, v.1, p.117-129.
- GLOVER, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing*, v.1, p.190-206.
- GLOVER, F.; KELLY, J.P.; LAGUNA, M. (1995). Genetic Algorithms and Tabu Search: Hybrids for Optimization. *Computers & Operations Research*, v.22, n.1, p.111-134.

- GOLDBERG, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison - Wesley.
- GUPTA, J.N.D. (1971). A Functional Heuristic for the Flow Shop Scheduling Problem. *Operational Research Quarterly*, v.22, p.39-47.
- HO, J.C.; CHANG, J. (1991). A New Heuristic for the n-job, m-machine Flow Shop Problem. *European Journal of Operational Research*, v.52, p.194-202.
- HOLLAND, J.H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, Mich.
- HUNDAL, T.S.; RAJGOPAL, J. (1988). An Extension of Palmer's Heuristic for the Flow Shop Scheduling Problem. *International Journal of Production Research*, v.26, p.1119-1124.
- IGNALL, E.; SCHRAGE, L.E. (1965). Application of Branch and Bound Technique to some Flow-Shop Problem. *Operations Research*, v.13, p.400-412.
- ISHIBUCHI, H.; MISAKI, S.; TANAKA, H. (1995). Modified Simulated Annealing Algorithms for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, v.81, p.388-398.
- JOHNSON, S.M. (1954). Optimal Two and Three Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly*, v.1, p.61-68.
- KIM, H.; NARA, K.; GEN, M. (1994). A Method for Maintenance Scheduling Using GA Combined with SA. *Computers & Industrial Engineering*, v.27, p.477-480.
- KIRKPATRICK, S.; GELATT, C.D. Jr.; VECCHI, M.P. (1983). Optimization by Simulated Annealing. *Science*, v.220, p.671-680.

- KURODA, M.; KAWADA, A. (1994). Improvement on the Computational Efficiency of Inverse Queueing Network Analysis. *Computers & Industrial Engineering*, v.27, p.421-424.
- MACCARTHY, B.L.; LIU, J. (1993). Addressing the Gap in Scheduling Research: a Review of Optimization and Heuristic Methods in Production Scheduling. *International Journal of Production Research*, v.31, n.1, p.59-79.
- MOCCELLIN, J.V. (1992). *Uma Contribuição à Programação de Operações em Sistemas de Produção Intermitente Flow Shop*. São Carlos. 126p. Tese (Livre-docência) - Escola de Engenharia de São Carlos, Universidade de São Paulo.
- MOCCELLIN, J.V. (1995). A New Heuristic Method for the Permutation Flow Shop Scheduling Problem. *Journal of the Operational Research Society*, v.46, p.883 - 886.
- MOCCELLIN, J.V.; NAGANO, M.S. (1998). Evaluating the Performance of Tabu Search Procedures for Flow Shop Sequencing. *Journal of the Operational Research Society*, v.49, p.1296 - 1302.
- MOSCATO, P. (1993). An introduction to population approaches for optimization and hierarchical objective function: A discussion on the role of tabu search. *Annals of Operations Research*, v.41, p.85-122 *apud* PIRLOT, M. (1996). General local search methods. *European Journal of Operational Research*, v.92, n.3, p.509.
- MOTA, W.L. (1996). *Análise Comparativa de Algoritmos Genéticos para o Problema de Programação de Operações Flow Shop Permutacional*. São Carlos. 125p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.

- MURATA, T., ISHIBUCHI, H; TANAKA, H. (1996). Genetic algorithms for flow shop problems. *Computers and Industrial Engineering*, v.30, n.4, p.1061-1071.
- NAGANO, M. S. (1995). *Novos procedimentos de Busca Tabu para o problema de programação de operações flow shop permutacional*. São Carlos. 117p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.
- NAWAZ, M.; ENSCORE Jr., E.E.; HAM, I. (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem. *Omega*, v.11, p.91-95.
- OGBU, F.A.; SMITH, D.K. (1990). The Application of the Simulated Annealing Algorithm to the Solution of the $n/m/C_{\max}$ Flowshop Problem. *Computers and Operations Research*, v.17, p.243-253.
- OSMAN, I.H.; POTTS, C.N. (1989). Simulated Annealing for Permutation Flow-Shop Scheduling. *Omega*, v.17, p.551-557.
- OW, P.S. (1985). Focused scheduling in proportionate flowshops. *Management Science*, v.70/3, p.852-869 *apud* DÍAZ, B.A. (1996). An SA/TS Mixture Algorithm for the Scheduling Tardiness Problem. *European Journal of Operational Research*, v.88, p.519.
- PALMER, D.S. (1965). Sequencing Jobs through a Multi-stage Process in the Minimum Total Time - A Quick Method of obtaining a Near Optimum. *Operational Research Quarterly*, v.16, p.101-107.
- PARK, M.W.; KIM, Y.D. (1998). A Systematic Procedure for Setting Parameters in Simulated Annealing Algorithms. *Computers & Operations Research*, v.25, p.207-217.

- PIRLOT, M. (1996). General local search methods. *European Journal of Operational Research*, v. 92, n.3, p. 493-511.
- REEVES, C.R. (1995). A Genetic Algorithm for Flowshop Sequencing. *Computers & Operations Research*, v.22, p.5-13.
- RESENDE, M.O.; SACOMANO, J.B. (1991). *Princípios dos Sistemas de Planejamento e Controle da Produção*. São Carlos, EESC/USP. /Apostila/ 224p.
- ROACH, A.; NAGI, R. (1996). A Hybrid GA-SA Algorithm for Just - In - Time Scheduling of Multi - Level Assemblies. *Computers & Industrial Engineering*, v.30, n.4, p.1047-1060.
- SELEN, W.J.; HOTT, D.D. (1986). A Mixed-Integer Goal Programming Formulation of the Standard Flow-Shop Scheduling Problem. *Journal of the Operational Research Society*, v.37, p.1121-1128.
- SILVA, E.C.S. (1994). *Kanban em Célula Piloto como Técnica Auxiliar do Planejamento e Controle da Produção: um Estudo de Caso em Fábrica de Médio Porte*. São Carlos. 124p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo.
- SLACK, N.; CHAMBERS, S.; HARLAND, C.; HARRISON, A.; JOHNSTON, R. (1996). *Administração da Produção*. São Paulo, Atlas.
- TAILLARD, E. (1990). Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, v.47, p.65-74.
- WIDMER, M.; HERTZ, A. (1989). A New Heuristic Method for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, v.41, p.186-193.

ZACCARELLI, S.B. (1986). *Programação e Controle da Produção*. São Paulo, Pioneira.

ZEGORDI, S.H.; ITOH, K.; ENKAWA, T. (1995). Minimizing Makespan for Flow Shop Scheduling by Combining Simulated Annealing with Sequencing Knowledge. *European Journal of Operational Research*, v.85, p.515-531.