DESENVOLVIMENTO DE UM MÉTODO METAHEURÍSTICO HÍBRIDO ALGORITMO GENÉTICO-BUSCA TABU PARA O PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES *FLOW-SHOP* PERMUTACIONAL

Angela Betânia Dias de Souza

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
2000

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

S729d
S719d

Souza, Angela Betânia Dias de
Desenvolvimento de um método metaheurístico híbrido Algoritmo Genético-Busca Tabu para o problema de programação de operações *Flow Shop* Permutacional./ Angela Betânia Dias de Souza. -- São Carlos, 2000.

Dissertação (Mestrado) -- Escola de Engenharia de São Carlos-Universidade de São Paulo, 2000.
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

Candidata: Bacharela **ANGELA BETANIA DIAS DE SOUZA**

Dissertação defendida e aprovada em 25-08-2000
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 **MARCELO SEIDO NAGANO**
(Faculdade de Economia e Administração - Universidade de São Paulo)

Prof. Associado **RENATO VAIRO BELHOT**
Coordenador do Programa de Pós-Graduação
em Engenharia de Produção

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

Memória

Amar o perdido
deixa confundido
este coração.

Nada pode o olvido
contra o sem sentido
apelo do Não.

As coisas tangíveis
tornam-se insensíveis
à palma da mão

Mas as coisas findas
muito mais que lindas,
essas ficarão.

DRUMMOND

Agradecimentos

- À Deus, pela vida.
- Ao Professor e amigo Dr. João Vitor Moccellin, pela orientação e compreensão fornecida durante todo o período de trabalho.
- À minha mãe Maria de Lourdes, pelo otimismo e coragem transmitidos durante toda a minha vida.
- Aos meus irmãos Helenildes e José Wilson, pela amizade, exemplo de honestidade e tranqüilidade diante dos obstáculos impostos pela vida.
- Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro.
- À todos os professores e funcionários da Escola de Engenharia de Produção - EESC-USP, pela colaboração.
- À todos os professores e amigos do Instituto de Ciências Matemáticas e Computação - ICMC-USP, pela formação e amizade.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

SOUZA, A. B. D. (2000). *Desenvolvimento de um Método Heurístico Híbrido combinando as Metaheurísticas Algoritmo Genético e Busca Tabu para a Solução do Problema de Programação de Operações Flow-Shop Permutacional.* São Carlos, 2000. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

Este trabalho trata do problema de programação de operações *Flow Shop* Permutacional. Este problema é considerado *NP-hard*, isto é, de difícil solução, por isso são encontrados na literatura diversos métodos heurísticos para a solução do problema. A vantagem de utilizar os métodos heurísticos é que eles garantem uma boa solução, e algumas vezes ótima, utilizando um tempo computacional relativamente pequeno. O Algoritmo Genético (AG) e a técnica de Busca Tabu (BT) são métodos heurísticos que melhoram soluções iniciais do problema a partir de procedimentos de busca no espaço de soluções. Uma idéia interessante que tem sido recentemente relatada na literatura, refere-se ao desenvolvimento de métodos heurísticos híbridos utilizando, por exemplo, as metaheurísticas AG e BT. O objetivo de combinar as técnicas metaheurísticas é obter um método híbrido que seja mais eficaz do que tais técnicas utilizadas isoladamente. Neste trabalho é apresentado um método heurístico híbrido Algoritmo Genético-Busca Tabu, chamado de *HBGATS*, para minimizar a duração total da programação F*low Shop* Permutacional. Para avaliar a eficácia da hibridização, compara-se o desempenho do método híbrido com os desempenhos dos métodos puros AG e BT que foram utilizados na hibridização. Os resultados obtidos na experimentação computacional são avaliados, concluindo-se sobre o desempenho do método híbrido *HBGATS* em relação aos métodos puros.

Palavras-chave: programação da produção, *Flow Shop* Permutacional, metaheurísticas híbridas.

ABSTRACT

SOUZA, A . B. D. (2000). Desenvolvimento de um Método Metaheurístico Híbrido combinando as Metaheurísticas Algoritmo Genético e Busca Tabu para a Solução do Problema de Programação de Operações Flow-Shop Permutacional. São Carlos, 2000. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

This work deals with the permutation Flow Shop Scheduling problem. This problem is considered NP-hard, that is, of dificult solution, this is why several heuristic methods are seen in the literature for the solution of such a problem. The advantage of using heuristics methods is that they provide a good solution and some times optimal, with a computational time relatively small. The Genetic Algorithm (GA) and the Tabu Search (TS) techniques are heuristics methods that improve the inicial solutions of the problem beginning with the procedure of search in the space of solutions. A promissing idea that shows up in the literature, refers to the development of hybrid heuristic methods, for example, the metaheuristics AG and BT. The objective of combining metaheuristic techniques is to obtain a hybrid method which is more effective than the ones used individually. In this work we present a hybrid heuristic method Genetic Algorithm-Tabu Search, for short HBGATS, for the minimal makespan flow shop sequencing problem. To evaluate the performance of the hybridsation, we compare the performance of the hybrid method with those of the pure AG and BT wich were used in the hybridsation. The results obtained in the computational experimentation are checked, and a conclusion got about the performance of the hybrid HBGATS method in relation to the pures.

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

CAPÍTULO 1

INTRODUÇÃO

1.1. Programação da Produção

A programação da produção é a alocação, no tempo, dos recursos disponíveis de produção, de tal forma que se estabeleçam as quantidades (e os lotes) a fabricar de cada produto, as datas de início e de término e os equipamentos que serão utilizados na fabricação desses produtos.

O lote de fabricação é simplemesnte a divisão da quantidade total a fabricar de um determinado produto, facilitando, com essa divisão, a fabricação a tempo de todos os produtos. O tamanho do lote depende da quantidade total a fabricar de cada produto, da posição dos estoques, dos custos de produção e do tipo de processo produtivo.

Deve-se ressaltar que quanto menor o lote, mais difícil será a programação da produção, exigindo mais rigor em seu controle, devido a ocorrência mais rápida da reposição dos estoques. Por isso, é necessário um ponto de equilíbrio entre Custos, Tamanho do Lote e Prazos (em função da demanda e da posição dos estoques).

Pelo fato da programação da produção e os estoques estarem relacionados diretamente com a demanda pelos produtos, é necessário administrar as variáveis controláveis (matérias-primas, níveis de estoques, carga-máquina, etc.), internas à empresa, e as variáveis não controláveis (demanda), externas à empresa, de modo a obter o melhor desempenho do sistema produtivo, atendendo às quantidades e prazos através da utilização eficiente dos recursos.

As datas de início de fabricação dos produtos são estabelecidas através da subtração de uma data de término (prazo de entrega), os tempos de execução (duração) e as tolerâncias da fabricação (indisponibilidade de equipamentos, por exemplo).

A fabricação de produtos quando não há estoque, será programada através da demanda imposta pelos clientes, sendo pequeno o grau de repetitividade das tarefas, com isso, é necessário maior controle, tornando a programação mais complexa. Já, a fabricação de produtos quando há estoque, será programada em três estágios (a orientação é para a reposição dos estoques):

i) determinar o prazo para o produto final entrar no estoque;
ii) determinar o início da fabricação do produto; e
iii)disponibilizar matéria-prima para a fabricação.

Algumas das medidas (critérios) para avaliar a eficiência da programação da produção são: nível de produtos acabados ou trabalhos em endamento (orientação interna); porcentagem de utilização dos recursos (externa e interna); porcentagem das ordens entregues no prazo ou antes dele (orientação externa); porcentagem de faltas nos estoques (orientação interna); quantidade de clientes perdidos (externa e interna); tempo parado por outros motivos: quebras, falta de ocupação, etc. (orientação interna); tempo que o cliente espera (orientação externa).

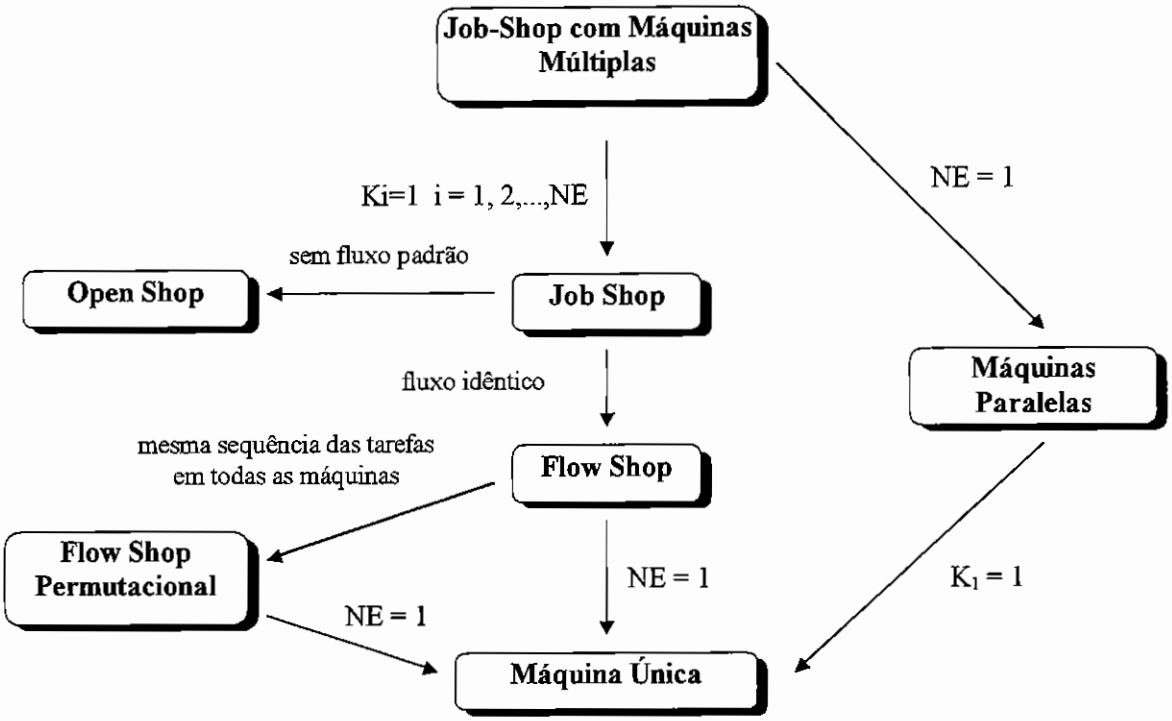
Os principais objetivos da programação da produção são: entregar os produtos fabricados nas datas compromissadas ou estabelecidas; distribuir a carga de trabalho de forma a obter máxima utilização dos recursos; garantir que toda a matéria-prima e compoenentes comprados estejam disponíveis quando forem solicitados pela fabricação; prever e evitar grande concentração de trabalho em poucas máquinas (gargalos de produção); prever a ociosidade da capacidade produtiva; estabelecer seqüências de produção que minimizem o tempo de equipamento sem trabalho.

1.2. Problemas de Programação de Operações em Máquinas

A programação da produção deve ser eficaz, utilizar todos os recursos tecnológicos disponíveis para produzir bens de maneira que satisfaça a seus consumidores. Por isso, as restrições tecnológicas e a medida de desempenho da programação devem ser especificadas. Nesse contexto, cada máquina desempenha um papel, ou seja, é responsável por uma ou mais atividades na linha de produção, essas atividades são chamadas de operações, e o produto final dessas operações em uma ou mais máquinas são chamadas de tarefas. Deve-se definir o roteiro de produção, o fluxo das operações das diversas tarefas nas máquinas com base nas restrições tecnológicas. Neste aspecto, os problemas de programação das operações nas máquinas podem ser classificados da seguinte forma:

a) *job-shop*: Cada tarefa tem sua própria seqüência de processamento nas máquinas.
b) *flow-shop*: Todas as tarefas têm a mesma seqüência de processamento nas máquinas. Em um *flow-shop* genérico, máquinas podem ser "puladas".
c) *open-shop*: Não há uma seqüência específica ou pré-estabelecida para as tarefas serem processadas nas máquinas.
d) *flow-shop* permutacional: É um *flow-shop* no qual em cada máquina a seqüência das tarefas é a mesma.
e) máquina única: Existe uma única máquina disponível.
f) máquinas paralelas: São disponíveis diversas máquinas, geralmente idênticas, para as mesmas operações.
g) *job-shop* com máquinas múltiplas: É um *job-shop* no qual existe um conjunto de máquinas paralelas em cada estágio.

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



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. (fonte : MACCARTHY & LIU (1993).

Dentre as diversas classes de programação de operações em máquinas, este trabalho está inserido no ambiente de programação de operações *Flow-Shop* Permutacional, no qual todas as tarefas têm a mesma seqüência de processamento em todas as máquinas.

Nas últimas quatro 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 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 qualidade da solução.

1.3. Objetivo do Trabalho

Este trabalho teve como objetivo desenvolver um método heurístico híbrido envolvendo as metaheurísticas Algoritmo Genético e Busca Tabu a solução do problema de Programação de Operações *Flow-Shop* Permutacional, de forma a ser mais eficaz, em termos da proximidade da solução ótima, do que os métodos metaheurísticos utilizados isoladamente.

CAPÍTULO 2

O PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES *FLOW-SHOP* PERMUTACIONAL

Conforme observado na figura 1, 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 10! (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.

Quando o número de tarefas é grande, o número de programações possíveis cresce fatorialmente e a busca por uma solução ótima torna-se complicada e muito demorada em termos computacionais, por isso os métodos heurísticos, por serem mais simples e

utilizarem esforços computacionais menores são muito utilizados para a solução desses problemas. 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**.

2.1. 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)).

2.2. 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.

CAPÍTULO 3

MÉTODOS HEURÍSTICOS PARA A SOLUÇÃO DO PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES *FLOW- SHOP* PERMUTACIONAL

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

• Seja **J** = $\{J_1, J_2, ..., J_i, ..., J_n\}$ 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 $\mathbf{J_i}$ na máquina **k** é $\mathbf{p_{ki}}$ (i = 1, 2, ..., n e k = 1, 2, ..., 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:

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

para: i = 1, 2, ..., 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, ..., 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:

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

Em cada um dos (m-1) estágios, 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 ≥ 4 máquinas, definindo para cada tarefa, o seguinte índice:

$$f(i) = \frac{A}{\min\limits_{1 \le k \le m-1} (p_{ki} + p_{k+1,i})}$$

onde: • i = 1, 2, ..., n e

• A = 1, se $p_{ki} \le p_{1i}$ 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:

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

para: i = 1, 2, ..., 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 soma de tempo de processamento nas m máquinas deve ter maior prioridade de programação em relação às tarefas com menor soma de tempo 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 a terceira maior soma do tempo 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 impar, 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:

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

para: i = 1, 2, ..., 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:

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

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 é descrita 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, ..., n;

• k = 1, 2, ..., (m-1);

• $i \neq j$;

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_i 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:

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

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:

$$UB\,X_{j+1}^{k+1} = max\,\{0,\ UB\,X_{j+1}^{k} + (p_{k,j+1} - p_{k+1,j})\}$$

onde:

- $UB\,X_{j+1}^{1} = 0$ (não existe tempo de espera entre tarefas sucessivas na primeira máquina);
- UB X_{j+1}^{k+1} : limitante superior de X_{j+1}^{k+1} ;
- UB X_{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 distintas de busca na vizinhança (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:

⇨ *Interchange Neighbourhood*: $f(n) = \dfrac{n(n-1)}{2}$, para $n \leq 40$ ou

$f(n) = 1780 - 1000\, e^{-0.03(n-40)}$, para $n > 40$

⇨ *Shift Neighbourhood:* $f(n) = (n-1)^2$, para $n \leq 30$ ou

$f(n) = 1741 - 900\, e^{-0.04(n-30)}$, 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 4.

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:

$$G(y_h) = \frac{h^K - (h-1)^K}{S^K}$$

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 $(y_1, y_2, ..., y_h)$.

Se todas as K soluções forem selecionadas de $(y_1, y_2, ..., y_h)$, então a solução candidata y é uma das $(y_1, y_2, ..., y_h)$ 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:

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

para: i = 1, ..., n.

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

- $D^{F}_{[i]} = \dfrac{\sum_{k=[1]}^{[i-1]} (AP_{[k]1} - AP_{[i]1})}{AP_{[i]1}}$ se $AP_{[i]1} \leq AP_{[i]2}$

- $D^{F}_{[i]} = \dfrac{\sum_{k=[1]}^{[i-1]} (AP_{[k]1} - AP_{[i]1})}{AP_{[i]2}}$ se $AP_{[i]1} > AP_{[i]2}$

- $D^{B}_{[i]} = \dfrac{\sum_{k=[i+1]}^{[n]} (AP_{[k]2} - AP_{[i]2})}{AP_{[i]2}}$ se $AP_{[i]2} \leq AP_{[i]1}$

- $D^{B}_{[i]} = \dfrac{\sum_{k=[i+1]}^{[n]} (AP_{[k]2} - AP_{[i]2})}{AP_{[i]1}}$ 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 freqüência 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*.

Os Capítulos 2 e 3 tiveram 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 4

MÉTODOS METAHEURÍSTICOS

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 Tabu, *Simulated Annealing* e Algoritmo Genético.

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 são também conhecidos pelo termo metaheurísticas.

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 a Busca Tabu e o Algoritmo Genético.

4.1. 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.

O Procedimento Genérico da Técnica de Busca Tabu

Passo 1 - Inicie com uma solução viável qualquer x_O e uma Lista Tabu vazia ($|T| = 0$).
Faça $x^* = x_O$, $c^* = c(x_O)$ e $K = 0$.
(x^* é a melhor solução encontrada até o momento e c^* é o valor da função-objetivo para tal solução).

Passo 2 - Em $M(x_K)$ escolha uma "mudança" m que minimize $c(m(x_K))$ e tal que não seja proibida pelos elementos da Lista Tabu T. Tal mudança pode ser determinada examinando-se parcial ou completamente a vizinhança $M(x_K)$. Faça $x_{K+1} = m(x_K)$.

Passo 3 - Se $c(x_{K+1}) < c^*$,
faça $c^* = c(x_{K+1})$ e $x^* = x_{K+1}$.

Passo 4 - Se $|T| = s$, remova o elemento mais antigo da Lista Tabu e acrescente o elemento t definido pela mudança m e pela solução x_{K+1}.
Incremente K de 1, ou seja, $K \leftarrow K+1$.

Passo 5 - Volte ao passo (2), se a "condição de parada" (por exemplo, solução ótima já encontrada ou K superior a um limite fixado), não for satisfeita.

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

4.2. *Simulated Annealing*

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 o menor nível de 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 e solução ótima).

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.

4.3. Algoritmo Genético

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.

O Princípio de Funcionamento do Algoritmo Genético

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.

A presente pesquisa 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.

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 sub-populações. 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 desse operador.

Pai 1: 0 1 0 | 1 1 ⇨ Filho 1: 0 1 0 | *0 1*

Pai 2: *1 1 0* | *0 1* ⇨ Filho 2: *1 1 0* | 1 1

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

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.

CAPÍTULO 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", isto é, percorrer os "vales" do espaço de soluções em busca de um ótimo global, não se restringindo às soluções locais (BT); aceitar momentaneamente soluções piores com uma certa probabilidade (SA) e diversificar a população através da aplicação de seus operadores, preservando certas características para os descendentes (AG), de tal forma que o procedimento resultante seja mais eficaz, garantindo uma solução mais próxima da solução ótima, 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 têm 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 freqüência 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*). Freqüência 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 explicitas 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 programa-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;

(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.

SUE (1998) estudou o problema de localização de pontos centrais (nós centrais ou trajetória central) com única alocação (*UHP-S*), usados para representar linhas aéreas (para facilitar o tráfego aéreo), porto de entregas de bagagens, sistemas de comunicações e outras redes de transportes. O objetivo desse estudo é determinar o número ideal de pontos centrais e proporcionar a rota ideal entre eles com baixo custo de transporte. O autor propõe um método híbrido das metaheurísticas Algoritmo Genético e Busca Tabu (*GATS*). Na implementação do Algoritmo Genético o cromossomo com representação binária (0-1) tem comprimento igual ao número de nós da rede considerada no problema, onde o número 1 representa que o nó é um ponto central e 0 ao contrário. A população inicial é gerada aleatoriamente e a seleção (pais) é feita via o método *Roulette Wheel*, o cruzamento foi feito pelo operador *one-point* e o operador de mutação é utilizado quando necessário para superar eventuais falhas após o cruzamento. Considerando uma seqüência σ com n nós, a vizinhança *shift* dessa seqüência é aquela em que remove-se o nó central da cidade (posição) **k** e o coloca na cidade **i**, no lugar de um nó não-central. E, a vizinhança *exchange* é a aquela onde permutam-se as cidades **i** e **k**, sendo $i \neq k$, pertencentes a nós (trajetórias) diferentes. Na implementação da Busca Tabu foram testadas as vizinhanças *shift* e *exchange* separadamente e a combinação entre elas. Os resultados computacionais mostraram que o método *GATS* é mais eficiente quanto utiliza somente a vizinhança *shift*

em relação a qualidade da solução encontrada em um tempo computacional relativamente pequeno comparado com *GATS-exchange* e *GATS-exchange e shift*. Os resultados computacionais demonstram que o Algoritmo Genético quando implementado sozinho obviamente encontra uma solução viável em tempo computacional menor que todos os outros métodos híbridos acima, mas a qualidade da solução encontrada é inferior.

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 da pesquisa que está sendo efetuada:

A presente pesquisa 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 Busca Tabu.

O método híbrido proposto, bem como o Algoritmo Genético puro que foi utilizado na sua concepção, são apresentados no próximo Capítulo.

CAPÍTULO 6

O MÉTODO METAHEURÍSTICO HÍBRIDO PROPOSTO *(HBGATS)*

Partindo-se do estudo realizado a respeito dos métodos metaheurísticos Algoritmo Genético e Busca Tabu e das hibridizações reportadas na literatura, propõe-se um método metaheurístico híbrido cuja base é composta por um Algoritmo Genético simples, associado a um método de Busca Tabu denominado ModFShop.TS5 (MOCCELLIN & NAGANO, 1998).

6.1. Parâmetros Quantitativos e Não-Quantitativos do Método Híbrido *HBGATS*

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). As mesmas soluções iniciais são utilizadas no Algoritmo Genético *puro* que deverá ser comparado com o híbrido. Por sua vez, no método Busca Tabu *puro* a ser utilizado na experimentação computacional, a solução inicial deverá ser a melhor dentre as fornecidas pelo NEH e FSHOPH.

Vizinhança :

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 .

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.

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 ModFShop.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 salientar que a mesma condição de parada será utilizada pelo Algoritmo Genético puro a ser comparado com o *HBGATS*.

Tamanho da Lista Tabu :

O tamanho da lista tabu é adotado igual a 7 (sete) conforme sugerido por GLOVER (1989). Na lista são registrados, a cada *movimento*, a tarefa que mudou de posição e a posição que ela ocupava na solução corrente anterior.

Operador de Reprodução :

No método híbrido e no Algoritmo Genético puro, 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.

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.

❶ *One Point Crossover* ou Operador de Cruzamento de UM CORTE

• PASSO 1: Escolha um ponto para se dividir cada pai, em duas subseqüências;

• PASSO 2: A solução filho é gerada através da união de duas subseqüências, uma de cada pai, na sua posição absoluta;

• PASSO 3: Para evitar a inviabilidade da solução filho, troca-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

❷ *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:

				≤ 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

Operador de Mutação :

Conforme já mencionado, 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 HBGATS*, uma vez que nesse método tal função é realizada pelo procedimento de Busca Tabu.

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 subseqüente;

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.

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.

E, 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.

6.2. O Algoritmo Genético Puro

(operadores de cruzamento onecutGA 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 ≤ TNbS(m,n) e $s_1 \neq s_2$:

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

s_1 ✖ s_2 → 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 *makespan*s

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

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

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

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

Se $f(s_1) <$ BM então

BS := seq[1]

BM := f(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.

6.3. O Método Híbrido *HBGATS*

Com o propósito de ilustrar a concepção do algoritmo HBGATS, os passos 1, 2 e 3 referem-se à "parte Algoritmo genético", enquanto que os *passos 4* e *5* (em *itálico*) correspondem à "parte Busca Tabu".

PASSO 1. Seleção dos pais: obtenha a seqüência inicial 1 (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}

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 ≤ TNbS(m,n) e $s_1 \neq s_2$:

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

s_1 ✖ s_2 → s_3 e s_4 (onde s_3 e s_4 são os descendentes)

Ordenar as seqüências em ordem crescente de 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(seq[1]) {makespan da primeira melhor seqüência}

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

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

Se $f(s_1)$ < BM então

BS := seq[1]

BM := f(seq[1])

PASSO 4. Gerar a vizinhança de inserção parcial aleatória de s_1 *(número de vizinhos* ***Nvp(S)*** *) e determinar a melhor seqüência vizinha* ***s**** *(a de menor makespan), utilizando a técnica de Busca Tabu.*

$s_1 := s^*$

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

Se $f(s_1) < BM$ *então*

$BS := s_1$

$BM := f(s_1)$

PASSO 5. Gerar a vizinhança de inserção parcial aleatória de s_2 *(número de vizinhos* ***Nvp(S)*** *) e determinar a melhor seqüência vizinha* ***s*** *(a de menor makespan), utilizando a técnica de Busca Tabu.*

$s_2 := s^{**}$

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

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 *HBGATS* 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 7

EXPERIMENTAÇÃO COMPUTACIONAL E ANÁLISE DOS RESULTADOS OBTIDOS

7.1. Experimentação Computacional

A experimentação computacional foi dividida em duas etapas:

❶ Primeiramente, testou-se o método híbrido Algoritmo Genético-Busca Tabu (*HBGATS)*, utilizando-se o operador de cruzamento de UM CORTE. Foram desenvolvidos 20 programas híbridos referentes aos 20 valores distintos da parcela p da vizinhança ($p \in$ {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}). Esses programas foram comparados com o método metaheurístico Algoritmo Genético puro (denominado onecutGA), utilizando-se o mesmo operador de cruzamento de UM CORTE, e com o método metaheurístico Busca Tabu puro (denominado ModFshop.TS5).

❷ Na segunda etapa, testou-se o método híbrido *HBGATS* utilizando-se o operador de cruzamento PMX (*Partially Matched Crossover*). Foram também desenvolvidos 20 programas híbridos para os 20 valores de p. Esses programas foram comparados com os métodos metaheurísticos Algoritmo Genético puro (pmxGA) e Busca Tabu (ModFshop.TS5), utilizando-se no pmxGA o mesmo operador de cruzamento PMX.

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

Todos os problemas foram gerados aleatoriamente, com tempos de processamento das tarefas em 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 a experimentação computacional, foi desenvolvido um *software* em linguagem Pascal chamado "Gerenciador de Problemas *Flow-Shop*", contendo os algoritmos onecutGA/pmxGA, ModFshop.TS, *HBGATS 1x* (com o operador de cruzamento UM CORTE) e *HBGATS pmx* (com o operador de cruzamento PMX).

O menu principal do *software* constitui-se das 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 *HBGATS1XGA*modFshop.TS5
  (ou sair do *HBGATS*pmxGAmodFshop.TS5)

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

i) máquinas agrupadas (opção "Calcular ESTATÍSTICAS em função do número de tarefas"); e

ii) máquinas não-agrupadas (opção "Calcular ESTATÍSTICAS de uma classe").

Abaixo são descritas as estatísticas utilizadas na Experimentação Computacional:

- ***Porcentagem de Sucesso*** - é calculada pelo número de vezes em que o método obteve a melhor solução, dividido pelo número total de problemas avaliados. 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.

- ***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. Deve-se ressaltar que todos os métodos comparados entre si têm a mesma solução inicial. A Melhoria Relativa é obtida pela expressão:

$$RI = \frac{M_I - M}{M_I}$$

onde:

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

- ***Tempo de Computação*** - é calculado apenas com o propósito de avaliação dos esforços computacionais, pois os métodos desenvolvidos neste trabalho foram projetados para não apresentarem tempos excessivos e com grandes diferenças para problemas da mesma classe (m,n).

Seguindo uma hierarquia, a estatística que deverá orientar a escolha do melhor método é a ***Porcentagem de Sucesso***. A Melhoria Relativa é a ela correlacionada e deverá ser usada para reiterar o desempenho quanto à estatística principal. Isto implica que,

quando um método apresenta valores superiores da porcentagem de sucesso, é estatisticamente esperado que a melhoria relativa também apresente valores superiores.

7.2. Análise dos Resultados Obtidos

Os resultados obtidos nas duas etapas da experimentação computacional foram analisados da seguinte forma:

(i) Analisou-se, primeiramente, os resultados das Porcentagens de Sucesso para **máquinas agrupadas** (m = 4, 7 e 10 máquinas e n = 20 a 100 tarefas). Foram comparadas as Porcentagens de Sucesso de cada um dos 20 algoritmos *HBGATS* (operadores de cruzamento UM CORTE e PMX) às Porcentagens de Sucesso obtidas pelos métodos puros Algoritmo Genético (operadores de cruzamento UM CORTE e PMX) e Busca Tabu ModFshop.TS5.

(ii) O método híbrido *HBGATS* que obteve o melhor desempenho (quando comparado ao Algoritmo Genético puro e ModFshop.TS5 na fase (i), foi analisado em termos de resultados estatísticos para **máquinas não-agrupadas**.

(iii) Nesta terceira fase foram utilizados os resultados referentes à Melhoria Relativa (estatística secundária), com o objetivo de escolher um ou mais métodos híbridos dentre aqueles que tiveram o melhor desempenho quanto às Porcentagens de Sucesso (fases (i) e (ii))

7.3. Método Híbrido *HBGATS* com Operador de Cruzamento de UM CORTE

A Tabela 2 ilustra os resultados obtidos na comparação das Porcentagens de Sucesso para **máquinas agrupadas** (m = 4, 7 e 10) em relação às diferentes parcelas **p** da vizinhança analisada.

Na elaboração da tabela acima citada, foi utilizada a seguinte estrutura:

- O elemento "**X**" indica que o método híbrido *HBGATS* obteve maior Porcentagem de Sucesso para uma determinada parcela da vizinhança analisada, quando comparado aos métodos Algoritmo Genético e Busca Tabu (ModFshop.TS5) puros;
- O elemento "•" indica que o método *HBGATS* apresentou a mesma Porcentagem de Sucesso obtida pelo ModFshop.TS5 (empate);
- O elemento "*" indica que o Algoritmo Genético puro obteve maior Porcentagem de Sucesso, quando comparado aos demais métodos e
- a célula **não preenchida** indica que o ModFshop.TS5 obteve maior Porcentagem de Sucesso, quando comparado aos demais métodos.

Nas tabelas que seguem, o código encontrado junto ao nome "*HBGATS*" indica o tipo de operador (UM CORTE (1x) ou PMX) utilizado no algoritmo híbrido e a parcela **p** da vizinhança analisada.

Exemplos:

- *HBGATS 1x05* ⇒ operador de cruzamento: UM CORTE e **p** = 0,05
- *HBGATS pmx10* ⇒ operador de cruzamento: PMX e **p** = 0,10

TABELA 2: Comparação das Porcentagens de Sucesso obtidas pelos métodos onecutGA, ModFshop.TS5 e *HBGATS* (UM CORTE), para máquinas agrupadas

Algoritmo	Numero de Tarefas								
	20	30	40	50	60	70	80	90	100
HBGATS 1x 05	X	X	X	X	X	X	X	X	X
HBGATS 1x 10	X	X	X	X	X	X	X	X	X
HBGATS 1x 15	X	X	X	X	X	X	X	X	X
HBGATS 1x 20	X	X	X	X	X	X	X	X	X
HBGATS 1x 25	X	X	X	X	X	X	X	X	X
HBGATS 1x 30	X		X	X	X	X	X	X	X
HBGATS 1x 35	X	X	X	X	X	X	X	X	X
HBGATS 1x 40	X		X	X	X	X	X	X	X
HBGATS 1x 45			X	X	X	X	X	X	X
HBGATS 1x 50			X	X	X	X		X	X
HBGATS 1x 55			X	X	X	•	X	X	X
HBGATS 1x 60	•		X	X	X	X	X	X	X
HBGATS 1x 65	X		X	X	X	•	X	X	X
HBGATS 1x 70		•	•	X	X	X	X	X	X
HBGATS 1x 75	X	X	X		X	X	X	X	X
HBGATS 1x 80	X		X	X	X	X	X	X	X
HBGATS 1x 85		•	X		X	X	X	X	X
HBGATS 1x 90	X		X	X	X	X	X	X	X
HBGATS 1x 95			X	X	X	X	X	X	X
HBGATS 1x 100			X	X		X	X	X	X

A Tabela 3, a seguir, ilustra o número de vitórias, empates e derrotas do método *HBGATS* (para máquinas agrupadas), em relação aos métodos puros analisados, em função do número de tarefas (9 classes).

TABELA 3: Número de vitórias, empates e derrotas do método *HBGATS* (UM CORTE referentes às Porcentagens de Sucesso, em função do número de tarefas

Algoritmo	NÚMERO DE VITÓRIAS	NÚMERO DE EMPATES	NÚMERO DE DERROTAS
HBGATS 1x 05	9	0	0
HBGATS 1x 10	9	0	0
HBGATS 1x 15	9	0	0
HBGATS 1x 20	9	0	0
HBGATS 1x 25	9	0	0
HBGATS 1x 30	8	0	1
HBGATS 1x 35	9	0	0
HBGATS 1x 40	8	0	1
HBGATS 1x 45	7	0	2
HBGATS 1x 50	6	0	3
HBGATS 1x 55	6	1	2
HBGATS 1x 60	7	1	1
HBGATS 1x 65	7	1	1
HBGATS 1x 70	6	2	1
HBGATS 1x 75	8	0	1
HBGATS 1x 80	8	0	1
HBGATS 1x 85	6	1	2
HBGATS 1x 90	8	0	1
HBGATS 1x 95	7	0	2
HBGATS 1x 100	6	0	3

Observando a Tabela 3, vemos que o método *HBGATS* apresenta desempenho superior (maior número de vitórias) aos métodos onecutGA e ModFshop.TS5. Os híbridos que mais se destacaram foram:

- *HBGATS 1x05*
- *HBGATS 1x 10*
- *HBGATS 1x15*
- *HBGATS 1x20*
- *HBGATS 1x25*
- *HBGATS 1x35*

As Tabelas 4 a 7 ilustram sucessivas comparações entre os métodos híbridos acima, a fim de buscar o método que apresenta melhor desempenho, em termos de Porcentagem de Sucesso e Melhoria Relativa para cada classe (m,n). Observemos que para cada parcela **p** da vizinhança analisada, existem 27 classes (m,n) de problemas.

TABELA 4: Comparação das Porcentagens de Sucesso para cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
HBGATS 1x 05	4	X	•	•	X	X				•
	7	X	X	X	X	X	X	X	X	X
	10	X		X	X	X	X	X	X	X
HBGATS 1x 10	4	X	•	•	X	X	•	X		•
	7	X	X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X	X
HBGATS 1x 15	4	X	X		•	X	•	X	X	•
	7	X	X	X	X	X	X	X	X	X
	10		X	X	X	X	X	X	X	X
HBGATS 1x 20	4	X	X	•	X	X		X	•	•
	7	X	X	X	X	X	X	X	X	X
	10	•	X	X	X	X	X	X	X	X
HBGATS 1x 25	4	X	•	•	X	•		X	X	•
	7	X	X	X	X	X	X	X	X	X
	10	X		X	X	X	X	X	X	X
HBGATS 1x 35	4	X	X	•	•			X		•
	7	•	X	X	X	X	X	X	X	X
	10		X	X	X	X	X	X	X	•

TABELA 5: Número de vitórias, empates e derrotas do método *HBGATS* (UM CORTE) considerando todas as classes (m,n), referentes às Porcentagens de Sucesso

Método Usado	Número de Vitórias	Número de Empates	Número de Derrotas	Total de Classes
HBGATS 1x 05	20	3	4	27
HBGATS 1x 10	**22**	**4**	**1**	**27**
HBGATS 1x 15	22	3	2	27
HBGATS 1x 20	**22**	**4**	**1**	**27**
HBGATS 1x 25	21	4	2	27
HBGATS 1x 35	18	5	4	27

A Tabela 5 mostra que os métodos híbridos *HBGATS 1x 10* e *HBGATS 1x 20* obtiveram os melhores desempenhos em relação às Porcentagens de Sucesso para máquinas não-grupadas, totalizando 22 vitórias, 4 empates e 1 derrota em comparação com os métodos 1xGA e ModFshop.TS5. A Tabela 6, a seguir, compara as Melhorias Relativas para cada classe (m,n).

TABELA 6: Comparação das Melhorias Relativas para cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
HBGATS 1x 10	4	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
HBGATS 1x 20	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

TABELA 7: Número de vitórias, empates e derrotas do método *HBGATS* (operador UM CORTE) considerando todas as classes (m,n), referentes às Melhorias Relativas (p = 0,10 e p = 0,20)

Método Híbrido	Número de Vitórias	Número de Empates	Número de Derrotas	Total de Classes
HBGATS 1x 10	23	3	1	27
HBGATS 1x 20	23	2	2	27

As Tabelas 6 e 7 referentes às Melhorias Relativas dos métodos *HBGATS 1x10* e *HBGATS 1x20* levam à conclusão que o *HBGATS 1x10* é o melhor dentre os métodos híbridos com o operador de UM CORTE (1 derrota).

A Tabela 8, a seguir, apresenta as Porcentagens de Sucesso para **máquinas agrupadas** do método *HBGATS 1x10* e dos métodos puros onecutGA e ModFshop.TS5. A Figura 4 ilustra tais resultados, mostrando a superioridade do método *HBGATS 1x10* em relação aos demais métodos puros.

TABELA 8: Porcentagens de Sucesso para máquinas agrupadas (onecutGA X ModFshop.TS5 X *HBGATS 1x 10)*

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
OnecutGA	20,0	23,3	23,3	18,3	28,3	35,0	28,3	35,0	43,3
ModFshop.TS5	68,3	65,0	63,3	66,7	65,0	76,7	63,3	70,0	71,7
HBGATS 1x 10	83,3	83,3	91,7	96,7	91,7	86,7	88,3	93,3	93,3

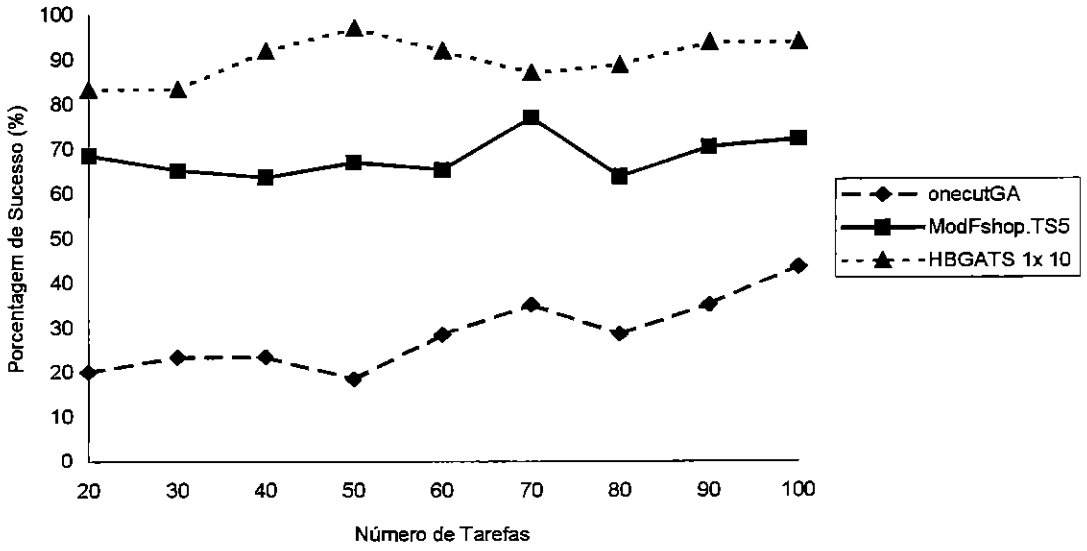


FIGURA 4: Porcentagens de Sucesso para máquinas agrupadas
(onecutGA X ModFshop.TS5 X *HBGATS 1x 10*)

7.4. Método Híbrido *HBGATS* com Operador de Cruzamento PMX

O método *HBGATS* com o operador de cruzamento PMX foi comparado com os métodos pmxGA (Algoritmo Genético com operador de cruzamento PMX) e ModFshop.TS5.

As Tabelas 9 e 10 mostram os resultados referentes à comparação das Porcentagens de Sucesso para **máquinas agrupadas**, obtidas pelos métodos pmxGA, ModFshop.TS5 e *HBGATS* (com o operador de cruzamento PMX) para os diversos valores da parcela **p** da vizinhança analisada. As tabelas seguintes seguem a mesma estruturação das tabelas anteriores.

TABELA 9: Comparação entre as Porcentagens de Sucesso obtidas pelos métodos pmxGA, ModFshop.TS5 e *HBGATS* (operador PMX), para máquinas agrupadas

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
HBGATS pmx 05	X	X	X	X	X	X	X	X	X
HBGATS pmx 10	X	X	X	X	X	X	X	X	X
HBGATS pmx 15	X	X	X	X	X	X	X	X	X
HBGATS pmx 20	X	X	X	X	X	X	X	X	X
HBGATS pmx 25	X	X	X	X	X	X	X	X	X
HBGATS pmx 30	X	X	X	X	X	X	X	X	X
HBGATS pmx 35	X	X	X	X	X	X	X	X	X
HBGATS pmx 40	X	X	X	X	X	X	X	X	X
HBGATS pmx 45	X	X	X	X	X	X	X	X	X
HBGATS pmx 50	X	X	X	X	X	X	X	X	X
HBGATS pmx 55	X	X	X	X	X	X	X	X	X
HBGATS pmx 60	X	X	X	X	X	X	X	X	X
HBGATS pmx 65	X	X	X	X	X	X	X	X	X
HBGATS pmx 70	X	X	X	X	X	X	X	X	X
HBGATS pmx 75	X	X	X	X	X	X	X	X	X
HBGATS pmx 80	X	X	X	X	X	X		X	X
HBGATS pmx 85	X	X	X	X	X	X		X	•
HBGATS pmx 90	X	X	X	X	X	X	X	X	X
HBGATS pmx 95	X	X	X	X	X	X		X	X
HBGATS pmx 100	X	X	X	X	X		X	X	X

TABELA 10: Número de vitórias, empates e derrotas do método *HBGATS* (operador PMX), referentes às Porcentagens de Sucesso, em função do número de tarefas

Algoritmo	NÚMERO DE VITÓRIAS	NÚMERO DE EMPATES	NÚMERO DE DERROTAS
HBGATS pmx 05	9	0	0
HBGATS pmx 10	9	0	0
HBGATS pmx 15	9	0	0
HBGATS pmx 20	9	0	0
HBGATS pmx 25	9	0	0
HBGATS pmx 30	9	0	0
HBGATS pmx 35	9	0	0
HBGATS pmx 40	9	0	0
HBGATS pmx 45	9	0	0
HBGATS pmx 50	9	0	0
HBGATS pmx 55	9	0	0
HBGATS pmx 60	9	0	0
HBGATS pmx 65	9	0	0
HBGATS pmx 70	9	0	0
HBGATS pmx 75	9	0	0
HBGATS pmx 80	8	0	1
HBGATS pmx 85	7	1	1
HBGATS pmx 90	9	0	0
HBGATS pmx 95	8	0	1
HBGATS pmx 100	8	0	1

Observando as Tabelas 9 e 10, vemos que o método híbrido *HBGATS* (com o operador PMX) supera (maior número de vitórias) os métodos puros pmxGA e ModFshop.TS5 para todos os valores da parcela p da vizinhança analisada. Os melhores valores da parcela da vizinhança, que obtiveram os melhores resultados são $\mathbf{p} \in \{0,05; 0,10; 0,15; 0,20; 0,25; 0,30; 0,35; 0,40; 0,45; 0,50; 0,55; 0,60; 0,65; 0,70, 0,75 \text{ e } 0,90\}$.

As Tabelas 11 e 12 mostram os resultados referentes à comparação das Porcentagens de Sucesso para **máquinas não-agrupadas** obtidas pelos métodos *HBGATS*, pmxGA e ModFshop.TS5 para cada valor de **p**, de acordo com o total de problemas em cada classe (m,n).

TABELA 11: Comparação das Porcentagens de Sucesso para cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
HBGATS pmx 05	4	X	•	•	X	X	X	X	X	•
	7	X	X	X	X	X	X	X	X	•
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 10	4	X	•		X	X	X		X	•
	7	X	X	X	X	X	X	X	X	
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 15	4	X	•	•	•	X	X	•	X	•
	7	X	X		X	X	X	X	X	
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 20	4	X		•	•	X	X	•		•
	7	X	X	X	X	X	X	X	X	
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 25	4	X			•	X	X	X	X	•
	7	X	X	X	X	X	X	X	X	
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 30	4	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
HBGATS pmx 35	4	X	X	X	•	•	X	X	X	•
	7		X	X	X	X	X	X	X	X
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 40	4	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
HBGATS pmx 45	4	X	X	X	•	•	X	X	•	•
	7	X	X	X	X	X	X	X	X	X
	10	X	X	X	X	X		X	X	X
HBGATS pmx 50	4	X	X	X	•	X	X	X	•	•
	7		•	X	X	X	X	X	X	X
	10	X	X		X	X	X	X	X	X
HBGATS pmx 55	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
HBGATS pmx 60	4	•		X	•	•	•	•	X	X
	7	X	X	X	•	X	X	X	X	X
	10	X	X	X	X	X	X	X	•	X
HBGATS pmx 65	4	X	•	•	X	X		•	X	X
	7	X	X	X	•		X	X	X	X
	10	X		X	X	X	X	X	X	X
HBGATS pmx 70	4	•		•	•	X		•	X	X
	7	X	X	X		X	X	X		
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 75	4	X	•	X	•	X	•	X	X	X
	7	X	X	X	X	X	X		X	X
	10	X	X	X	X	X	X	X	X	X
HBGATS pmx 90	4	X	X	X	•	•	•	X	•	•
	7		X	X	•	X		X	X	X
	10	X	X	X	X	X	X		X	X

TABELA 12: Número de vitórias, empates e derrotas do método *HBGATS* (operador PMX) considerando todas as classes (m,n), referentes às Porcentagens de Sucesso

Método Híbrido	Número de Vitórias	Número de Empates	Número de Derrotas	Total de Classes
HBGATS pmx 05	**23**	**4**	**0**	**27**
HBGATS pmx 10	22	2	3	27
HBGATS pmx 15	20	5	2	27
HBGATS pmx 20	20	4	3	27
HBGATS pmx 25	22	2	3	27
HBGATS pmx 30	**23**	**4**	**0**	**27**
HBGATS pmx 35	23	3	1	27
HBGATS pmx 40	**23**	**4**	**0**	**27**
HBGATS pmx 45	22	4	1	27
HBGATS pmx 50	21	4	2	27
HBGATS pmx 55	24	1	2	27
HBGATS pmx 60	19	7	1	27
HBGATS pmx 65	20	4	3	27
HBGATS pmx 70	18	4	5	27
HBGATS pmx 75	23	3	1	27
HBGATS pmx 90	18	6	3	27

Através das Tabelas 11 e 12, em relação ao número de vitórias, empates e derrotas referentes às Porcentagens de Sucesso dos métodos *HBGATS* (com o operador de cruzamento PMX), ModFshop.TS5 e pmxGA, podemos verificar que os métodos *HBGATS pmx 05, HBGATS pmx 30* e *HBGATS pmx 40* obtiveram os melhores desempenhos entre todos os demais métodos, totalizando 23 vitórias, 4 empates e 0 derrotas. Observa-se que o método *HBGATS pmx 55* obteve um número de vitórias (24) maior que os demais métodos, mas obteve 2 derrotas e 1 empate, por isso ele perde para os demais métodos citados acima. Abaixo, é feita a comparação das Melhorias Relativas dos métodos *HBGATS pmx 05, HBGATS pmx 30* e *HBGATS pmx 40.*

TABELA 13: Comparação das Melhorias Relativas para cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
HBGATS pmx 05	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
HBGATS pmx 30	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
HBGATS pmx 40	4	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

TABELA 14: Número de vitórias, empates e derrotas do método *HBGATS* (operador PMX), considerando todas as classes (m,n), referentes às Melhorias Relativas (p = 0,05, p = 0,30 e p = 0,40)

Método Híbrido	Número de Vitórias	Número de Empates	Número de Derrotas	Total de Classes
HBGATS pmx 05	**24**	**3**	**0**	**27**
HBGATS pmx 30	**24**	**3**	**0**	**27**
HBGATS pmx 40	23	2	2	27

As Tabelas 13 e 14, referentes às Melhorias Relativas, mostram que os métodos híbridos *HBGATS pmx 05* e *HBGATS pmx 30* são superiores ao método *HBGATS pmx 40*, pois obtiveram, igualmente, o menor número de derrotas (0) e o maior número de vitórias (24).

A Tabela 15 mostra as Porcentagens de Sucesso obtidas pelos métodos híbridos *HBGATS pmx 05* e *HBGATS pmx 30*.

TABELA 15: Porcentagens de Sucesso para cada classe (m,n) (p =0,05 e p = 0,30)

Operador	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
HBGATS pmx 05	4	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
	7	95,0	80,0	85,0	85,0	95,0	90,0	100,0	95,0	85,0
	10	75,0	85,0	90,0	100,0	90,0	90,0	95,0	95,0	100,0
HBGATS pmx 30	4	100,0	100,0	100,0	100,0	95,0	100,0	100,0	100,0	100,0
	7	75,0	85,0	95,0	85,0	95,0	90,0	95,0	90,0	95,0
	10	75,0	80,0	80,0	90,0	100,0	80,0	80,0	95,0	90,0

Na Tabela 16, a seguir, são comparados os valores referentes às Porcentagens de Sucesso obtidos pelos métodos *HBGATS pmx 05* e *HBGATS pmx 30*, onde usaremos os símbolos:

- " X " para indicar que o método *HBGATS pmx 05* obteve maior porcentagem de sucesso que o método *HBGATS pmx 30* ;
- " " (vazio) para indicar que o método *HBGATS pmx 30* obteve maior porcentagem de sucesso que o método *HBGATS pmx 05* ; e
- " • " para indicar que os dois métodos obtiveram a mesma porcentagem de sucesso.

TABELA 16: Comparação das Porcentagens de Sucesso obtidas pelos métodos *HBGATS pmx 05* X *HBGATS pmx 30*, para cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas								
		20	30	40	50	60	70	80	90	100
HBGATS (PMX)	4	•	•	•	•	X	•	•	•	•
	7	X			•	•	•	X	X	
	10	•	X	X	X		X	X	•	X

TABELA 17: Número de vitórias, empates e derrotas do método *HBGATS pmx 05* X *HBGATS pmx 30*, considerando todas as classes (m,n), referentes aos valores das Porcentagens de Sucesso

Método Obtido	Número de Vitórias	Número de Empates	Número de Derrotas	Total de Classes
HBGATS pmx 05	10	13	4	27
HBGATS pmx 30	4	13	10	27

O método *HBGATS pmx 05* obteve o menor número de derrotas (4) e o maior número de vitórias (10) em relação aos valores obtidos pelas Porcentagens de Sucesso em comparação ao método *HBGATS pmx* 30, e conseqüentemente ele pode ser considerado o melhor em relação a todos os métodos híbridos com o operador de cruzamento PMX.

A Figura 5, a seguir, ilustra as Porcentagens de Sucesso para máquinas agrupadas dos métodos pmxGA, ModFshopTS5 e *HBGATS pmx 05*.

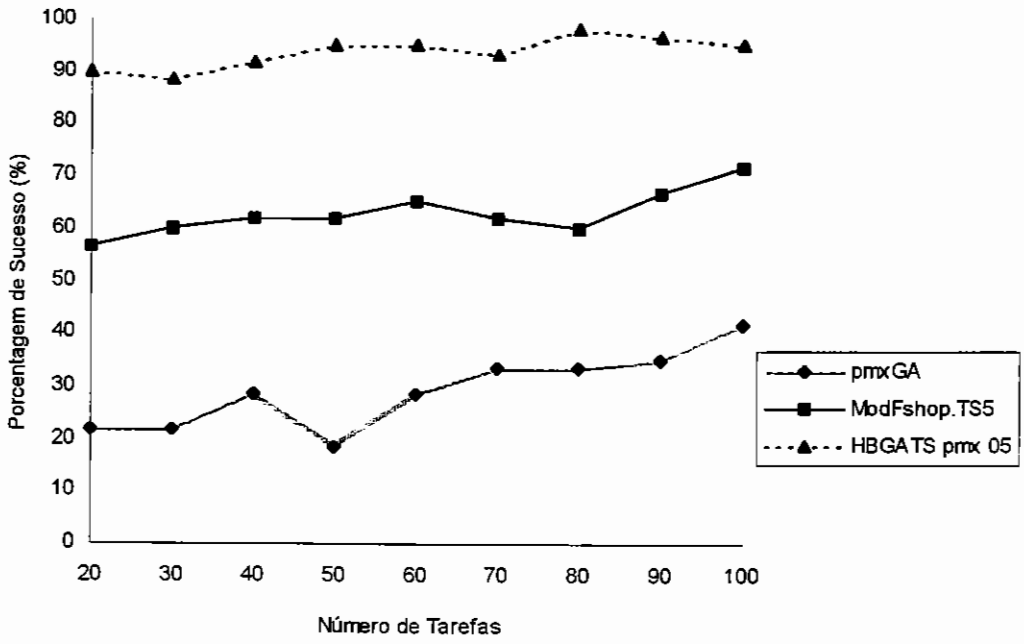


FIGURA 5: Porcentagens de Sucesso para máquinas agrupadas (pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

7.5. Escolha do Melhor Método Híbrido

Os híbridos *HBGATS 1x10* (com o operador UM CORTE) e *HBGATS pmx 05* (com o operador PMX) se destacaram entre todos os métodos híbridos e/ou puros analisados. As Tabelas 18 e 19 a seguir, comparam as Porcentagens de Sucesso para máquinas agrupadas obtidas por esses dois métodos híbridos, onde na Tabela 19 serão utilizados os símbolos:

- " X " para indicar que o método *HBGATS pmx 05* obteve maior porcentagem de sucesso que o método *HBGATS 1x10* ;
- " " (vazio) para indicar que o método *HBGATS 1x10* obteve maior porcentagem de sucesso que o método *HBGATS pmx 05* ; e
- " • " para indicar que os dois métodos obtiveram a mesma porcentagem de sucesso.

TABELA 18: Porcentagens de Sucesso dos métodos *HBGATS 1x10* (operador UM CORTE) X *HBGATS pmx 05* (operador PMX) para máquinas agrupadas

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
HBGATS 1x10	83,3	83,3	91,7	96,7	91,7	86,7	88,3	93,3	93,3
HBGATS pmx 05	90,0	88,3	91,7	95,0	95,0	93,3	98,3	96,7	95,0

TABELA 19: Comparação das Porcentagens de Sucesso dos métodos *HBGATS 1x10* (operador UM CORTE) X *HBGATS pmx 05* (operador PMX) para máquinas agrupadas

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
HBGATS	X	X	•		X	X	X	X	X

É nítido o ótimo desempenho do método *HBGATS pmx 05*, obtendo 7 vitórias, um empate e apenas uma derrota em relação ao método *HBGATS 1x10*.

A Figura 6, a seguir, ilustra as Porcentagens de Sucesso para máquinas agrupadas dos métodos híbridos *HBGATS 1x 10* e *HBGATS pmx 05*.

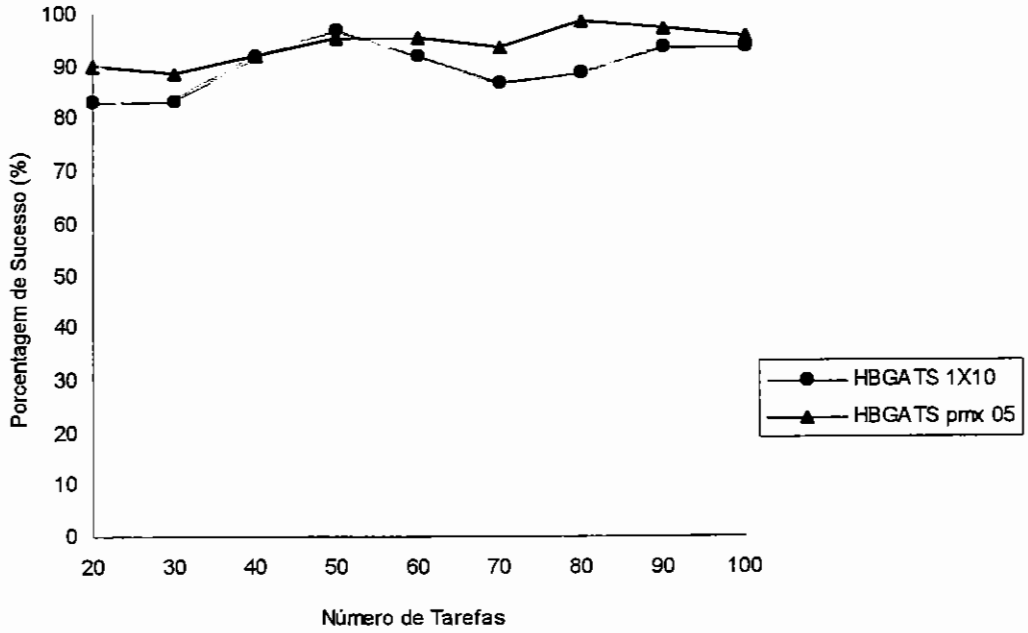


FIGURA 6: Porcentagens de Sucesso para máquinas agrupadas
(HBGATS 1X10 X HBGATS pmx 05)

A seguir, nas Tabelas 20 a 22, são comparadas as Porcentagens de Sucesso de ambos os métodos, considerando cada classe (m,n). Na Tabela 23 são utilizados os mesmos símbolos da Tabela 21.

TABELA 20: Porcentagens de Sucesso dos métodos *HBGATS 1x10* (operador UM CORTE) X *HBGATS pmx 05* (operador PMX), considerando cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas 20	30	40	50	60	70	80	90	100
HBGATS 1x10	4	95,0	95,0	100,0	100,0	100,0	95,0	90,0	95,0	100,0
	7	85,0	80,0	100,0	85,0	85,0	100,0	90,0	85,0	90,0
	10	65,0	45,0	85,0	95,0	95,0	100,0	80,0	85,0	95,0
HBGATS pmx 05	4	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
	7	95,0	80,0	85,0	85,0	95,0	90,0	100,0	95,0	85,0
	10	75,0	85,0	90,0	100,0	90,0	90,0	95,0	95,0	100,0

TABELA 21: Comparação das Porcentagens de Sucesso obtidas pelos métodos *HBGATS 1x10* X *HBGATS pmx 05*, considerando cada classe (m,n)

Algoritmo	Número de Máquinas	Número de Tarefas 20	30	40	50	60	70	80	90	100
HBGATS	4	X	X	•	•	•	X	X	X	•
	7	X	•		•	X		X	X	
	10	X	X	X	X			X	X	X

TABELA 22: Número de vitórias, empates e derrotas do método *HBGATS 1x10* X *HBGATS pmx 05* considerando todas as classes (m,n), referentes às Porcentagens de Sucesso

Método Utilizado	Número de Vitórias	Número de Empates	Número de Derrotas	Total de Classes
HBGATS 1x10	5	6	16	27
HBGATS pmx 05	16	6	5	27

O método *HBGATS pmx 05* (com o operador de cruzamento PMX), obteve desempenho superior ao *HBGATS 1x 10* (com o operador de cruzamento UM CORTE), pois obteve maior número de vitórias (16), referentes às Porcentagens de Sucesso para todas as classes (m,n) de problemas.

Em função da análise dos resultados obtidos na experimentação computacional, verifica-se a superioridade do método *HBGATS pmx 05* (com o operador de cruzamento PMX e p = 0,05), em relação aos demais métodos híbridos e/ou puros analisados no presente trabalho.

As Figuras 7 a 9, a seguir, ilustram as Porcentagens de Sucesso para máquinas não-grupadas dos métodos pmxGA, ModFshopTS5 e *HBGATS pmx 05*.

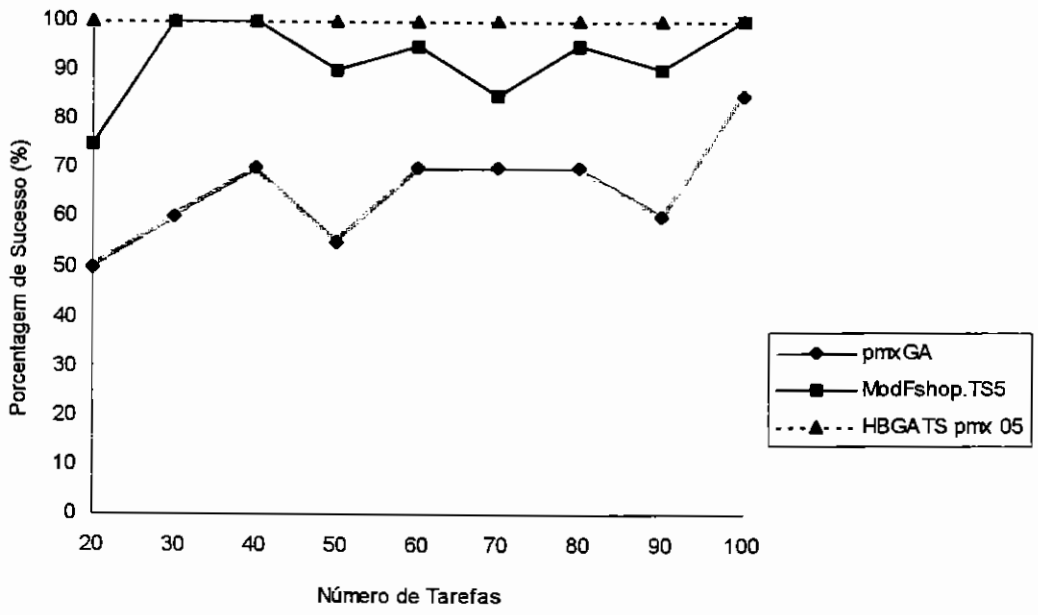


FIGURA 7: Porcentagens de Sucesso para máquinas não-agrupadas (m = 4) (pmxGA X ModFshop.TS5 X *HBGATS pmx 05)*

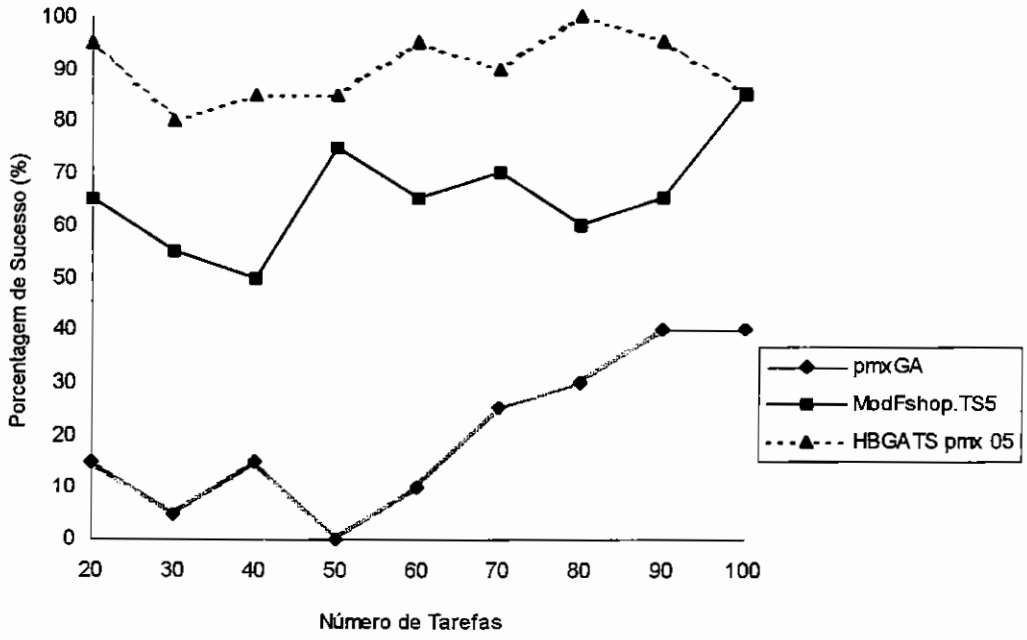


FIGURA 8 :Porcentagens de Sucesso para máquinas não-agrupadas (m = 7)
(pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

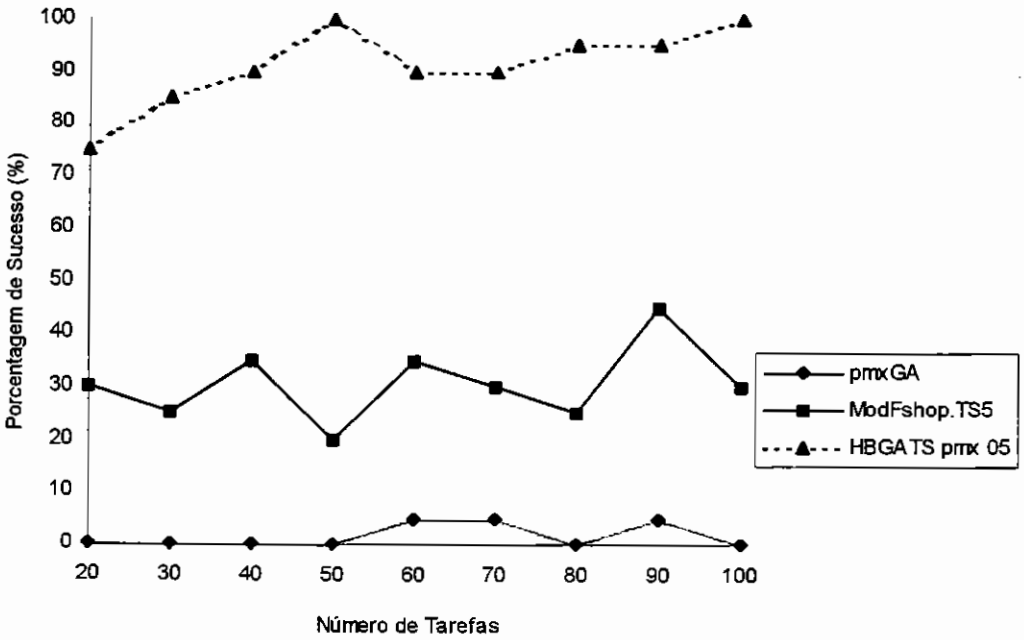


FIGURA 9: Porcentagens de Sucesso para máquinas não-agrupadas (m = 10)
(pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

Para m = 4, os métodos *HBGATS pmx 05* e ModFshopTS5 obtiveram as mesmas porcentagens de sucesso para n = 30, 40 e 100, e para os outros valores de n, o híbrido *HBGATS pmx 05* obteve desempenho superior aos demais métodos. As Figuras 8 a 9, para m = 7 e 10, e máquinas agrupadas, respectivamente, mostram a superioridade do método *HBGATS pmx 05* em relação aos demais métodos.

O bom desempenho do método *HBGATS pmx 05* também pode ser observado através das Melhorias Relativas, conforme mostram as Figuras 10 a 13.

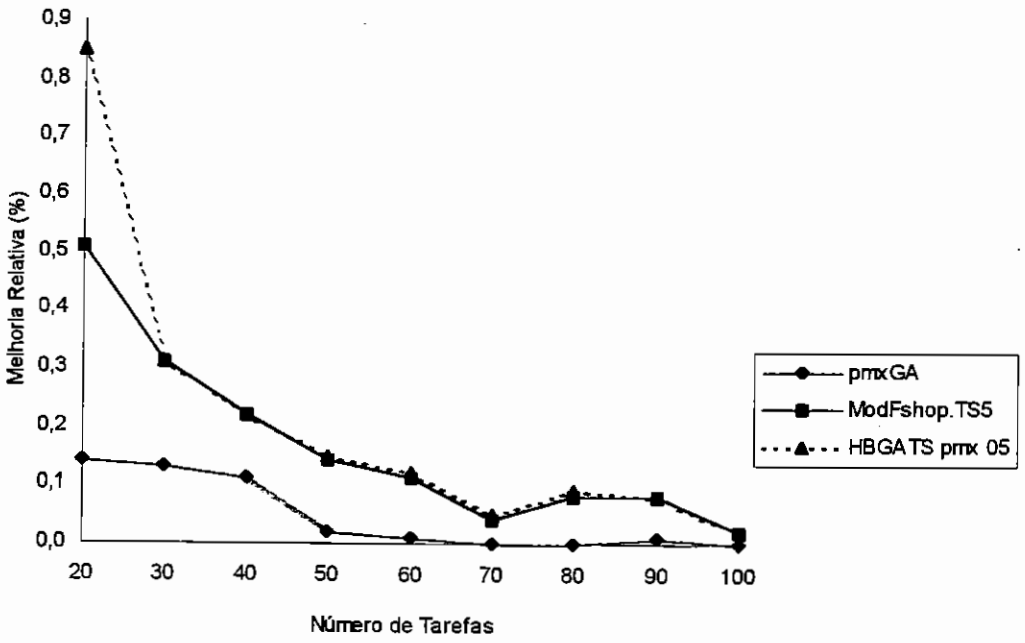


FIGURA 10: Melhorias Relativas (em porcentagem) para máquinas não-grupadas (m = 4) (pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

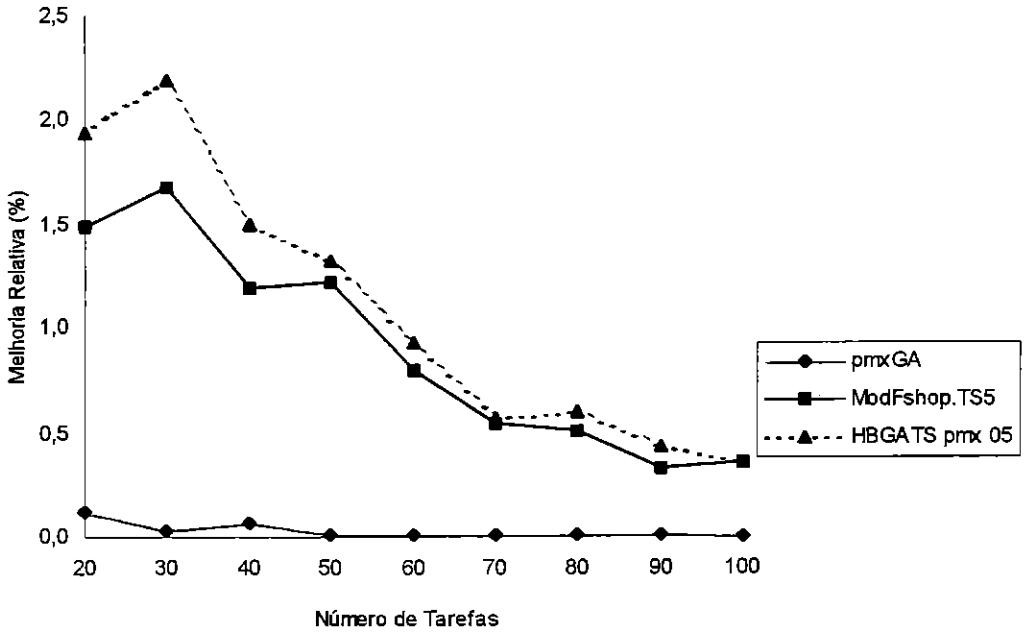


FIGURA 11: Melhorias Relativas (em porcentagem) para máquinas não-agrupadas (m = 7) (pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

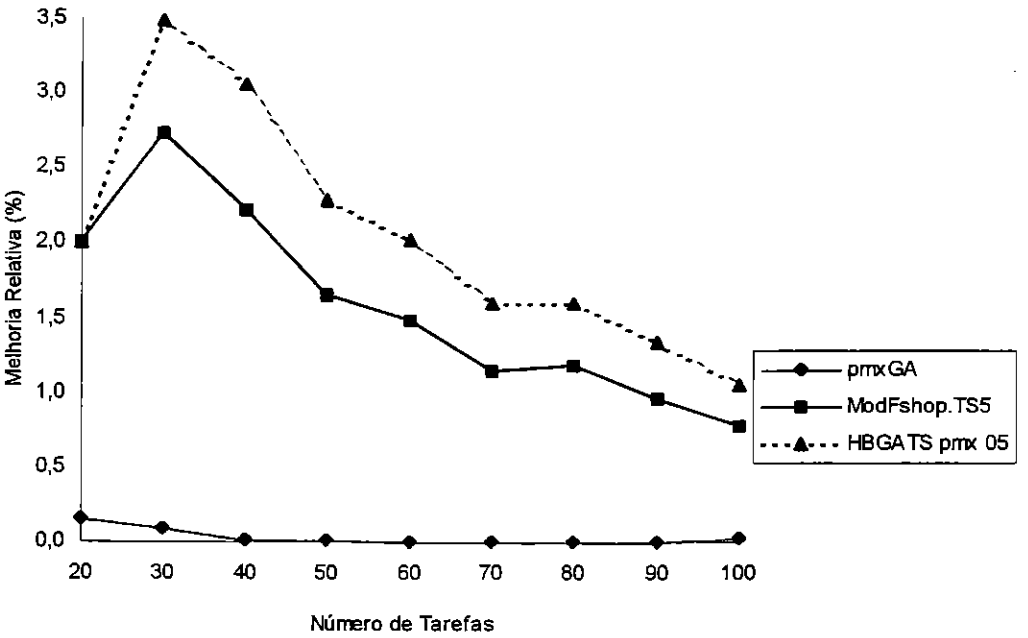


FIGURA 12: Melhorias Relativas (em porcentagem) para máquinas não-agrupadas (m =10) (pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

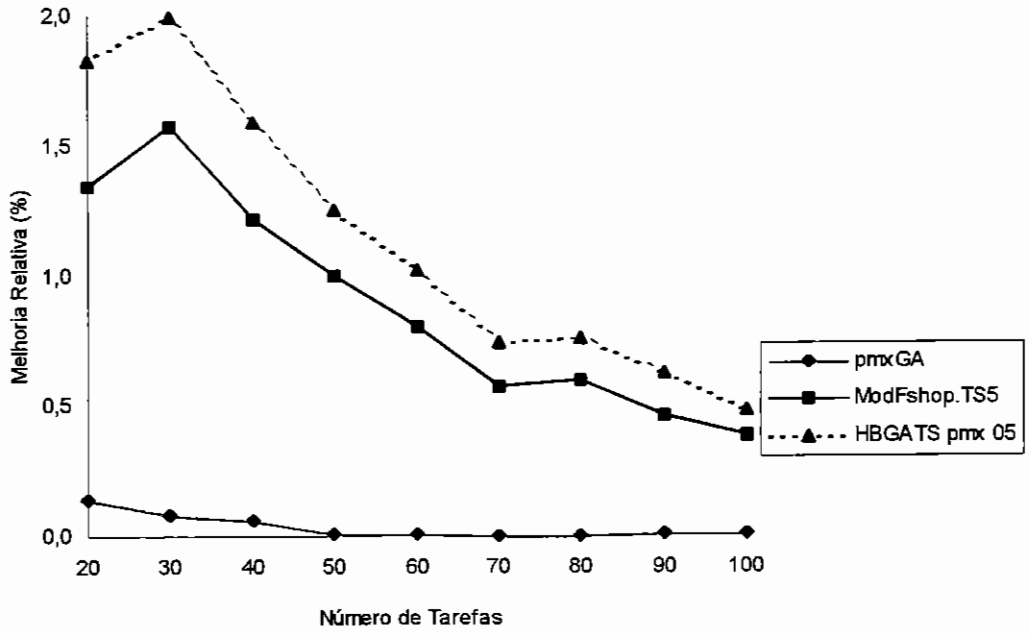


FIGURA 13: Melhorias Relativas (em porcentagem) para máquinas agrupadas (pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

A Figura 10, m = 4, mostra que para n ≥ 30 tarefas, os métodos *HBGATS pmx 05* e ModFshop.TS5 apresentaram comportamentos semelhantes em relação às Melhorias Relativas. Nas Figuras 11 a 13, m = 7 e 10, e máquinas agrupadas, respectivamente, é nítida a superioridade do método *HBGATS pmx 05*. Podemos também observar que em todos os casos os métodos *HBGATS pmx 05* e ModFshop.TS5 superam o desempenho do pmxGA.

As Figuras 14 a 17 comparam os Tempos de Computação (em segundos) para máquinas não-agrupadas (m = 4, 7 e 10) e agrupadas dos métodos pmxGA, ModFshop.TS5 e *HBGATS pmx 05*.

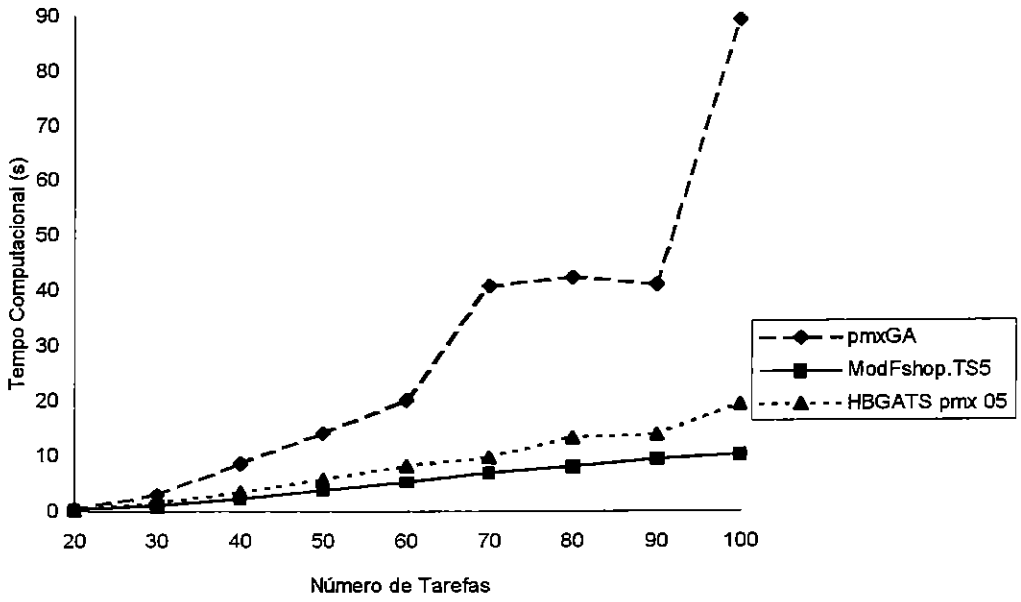


FIGURA 14: Tempos de Computação para máquinas não-agrupadas (m = 4)
(pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

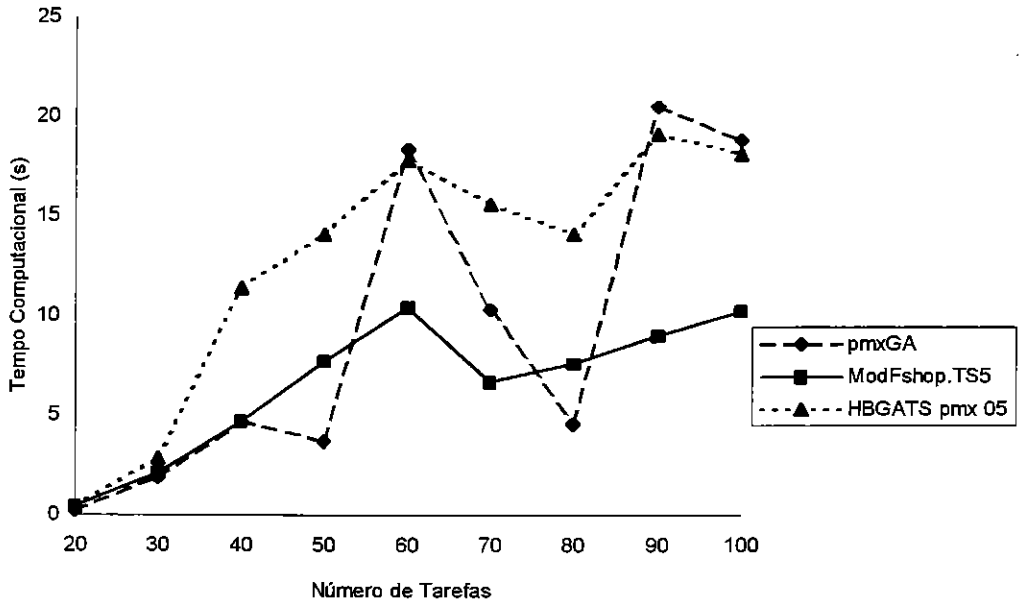


FIGURA 15: Tempos de Computação para máquinas não agrupadas (m = 7)
(pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

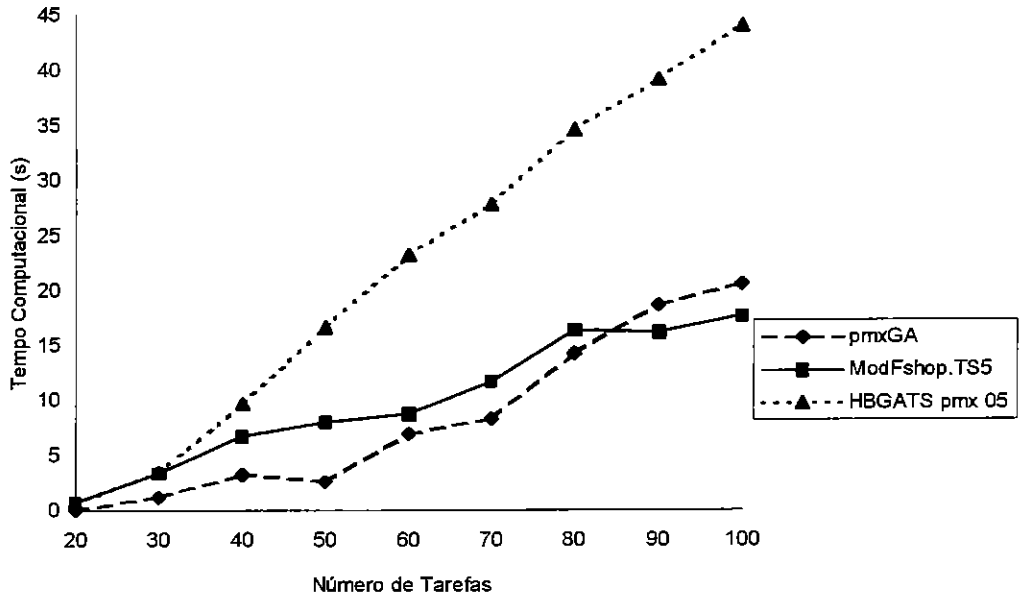


FIGURA 16: Tempos de Computação para máquinas não agrupadas (m = 10)
(pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

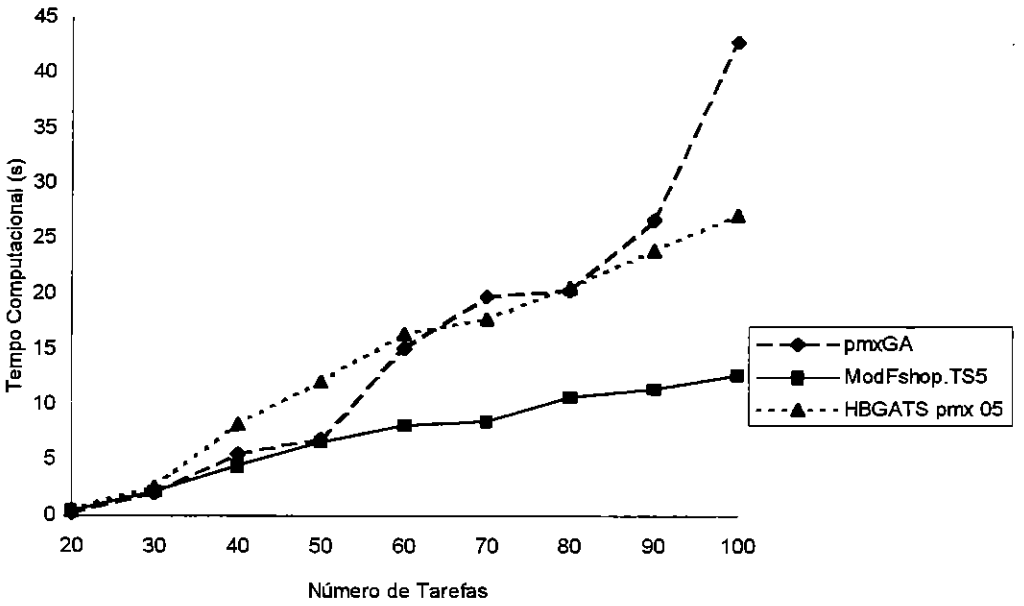


FIGURA 17: Tempos de Computação para máquinas agrupadas
(pmxGA X ModFshop.TS5 X *HBGATS pmx 05*)

A Figura 14 (m = 4) mostra que o método *HBGATS pmx 05* exigiu um tempo computacional superior ao ModFshop.TS5, e o pmxGA exigiu um alto esforço computacional em relação aos demais métodos. É importante ressaltar que o método puro pmxGA e o método híbrido *HBGATS pmx 05* podem encerrar a busca por melhores soluções quando as seqüências sobre as quais for aplicado o operador de cruzamento forem iguais (condição de parada secundária). Neste caso, as seqüências iniciais foram muito diferentes entre si, gerando seqüências-filhos, através do operador de cruzamento, também muito diferentes, exigindo, portanto, uma alta carga computacional em busca de melhores soluções. O método híbrido *HBGATS pmx 05*, analisou um número menor de seqüências em comparação ao pmxGA, devido à utilização da técnica de Busca Tabu, onde a busca por melhores soluções é feita na vizinhança de uma boa solução. Na Figura 15 (m = 7), o *HBGATS pmx 05* exigiu um esforço computacional superior aos demais métodos em quase todos os números de tarefas, exceto para n = 60, 90 e 100, onde o método pmxGA teve um tempo computacional superior aos demais métodos, devido à geração de seqüências muito diferentes entre si, para esses valores de n. A Figura 16 (m = 10) mostra que o método *HBGATS pmx 05* exigiu um esforço computacional bem superior aos demais métodos.

Podemos verificar, através da Figura 17, que o método híbrido *HBGATS pmx 05*, exigiu, em média, um esforço computacional superior aos demais métodos em quase todos os números de tarefas, exceto para n = 70, 90 e 100, onde o método pmxGA teve um tempo médio computacional superior aos demais métodos, pelo motivo já mencionado.

Os resultados obtidos na experimentação computacional para os métodos pmxGA, ModFshop.TS5 e *HBGATS pmx 05* relativos a Porcentagens de Sucesso, Melhorias Relativas e Tempos de Computação para máquinas não-agrupadas e agrupadas estão tabelados no **Anexo 1**.

CAPÍTULO 8

CONCLUSÕES

A finalidade do presente trabalho foi obter um método metaheurístico híbrido Algoritmo Genético – Busca Tabu para o problema de Programação de Operações *Flow-Shop* Permutacional, objetivando a minimização do *makespan*. O método desenvolvido *HBGATS* utilizou-se das características básicas do Algoritmo Genético simples (seleção dos pais e aplicação do operador de cruzamento) e da Busca Tabu.

Foram desenvolvidos e testados 540 algoritmos híbridos utilizando o operador de cruzamento UM CORTE e 540 algoritmos híbridos utilizando o operador de cruzamento PMX, referentes às 27 classes de problemas, combinando-se m máquinas e n tarefas (m = 4, 7 e 10 e n = 20, 30, 40, 50, 60, 70, 80, 90 e 100), e a parcela **p** ($p \in [0,05; 1,00]$) da vizinhança analisada. Nas experimentações computacionais foram comparados os algoritmos híbridos com Algoritmos Genéticos puros (onecutGA/pmxGA) e Busca Tabu puro (ModFshop.TS5).

8.1. O Método Híbrido *HBGATS* (operador UM CORTE/PMX)

As Tabelas 2 a 7 referentes às Porcentagens de Sucesso e Melhorias Relativas obtidas pelos métodos puros e híbridos, mostram a superioridade do método híbrido *HBGATS 1x10* (operador de cruzamento UM CORTE e p = 0,10) em relação aos demais métodos analisados com o operador de cruzamento UM CORTE.

As Tabelas 9 a 17 referentes às Porcentagens de Sucesso e Melhorias Relativas obtidas pelos métodos puros e híbridos, mostram a superioridade do método híbrido

HBGATS pmx 05 (operador de cruzamento PMX e p = 0,05) em relação aos demais métodos analisados com o operador de cruzamento PMX.

8.1.1. Superioridade do Operador de Cruzamento PMX

As Tabelas 2 e 9 referentes às Comparações das Porcentagens de Sucesso para máquinas agrupadas obtidas pelos métodos puros e híbridos utilizando os operadores de cruzamento UM CORTE e PMX, respectivamente, mostram a superioridade do método híbrido ***HBGATS*** utilizando o operador de cruzamento **PMX**. Isto ocorreu pelo fato do método *HBGATS* utilizar uma condição secundária de parada, que encerra a busca de melhores soluções quando as seqüências sobre as quais for aplicado o operador de cruzamento forem iguais, e o operador de UM CORTE, utilizado freqüentemente na literatura em Algoritmos Genéticos, apesar de ser muito eficiente, gerou seqüências-pais idênticas com alta freqüência e muitas vezes após um número pequeno de iterações, por isso o método *HBGATS* com o operador de cruzamento PMX obteve desempenho superior ao *HBGATS* com o operador de cruzamento UM CORTE, em relação às Porcentagens de Sucesso, e conseqüentemente às Melhorias Relativas.

8.1.2. O Melhor Método Híbrido

As Tabelas 18 a 22, mostram que o método híbrido ***HBGATS pmx 05*** obteve desempenho superior ao *HBGATS 1x10* e, portanto, aos demais métodos híbridos (com os operadores de cruzamento UM CORTE e PMX) e puros ModFshopTS5, onecutGA e pmxGA.

8.1.3. Parcela da Vizinhança Analisada

Através das Tabelas 2 e 3, verifica-se que o método *HBGATS* com o operador de cruzamento UM CORTE, em relação à parcela **p** da vizinhança analisada, apresentou melhor desempenho para valores relativamente baixos do parâmetro **p**. As Tabelas 9 e 10

mostram que o método *HBGATS* com o operador de cruzamento PMX obteve um ótimo desempenho em praticamente todos os valores do parâmetro **p**.

8.2. Desempenho do Algoritmo Genético Puro

O desempenho do Algoritmo Genético puro (onecutGA/pmxGA) se mostrou sempre inferior aos métodos ModFshop.TS5 e *HBGATS* (com os operadores de cruzamento UM CORTE e PMX), em relação às estatísticas de Porcentagens de Sucesso e Melhorias Relativas para máquinas agrupadas e não-agrupadas. A superioridade dos métodos ModFshop.TS5 e *HBGATS* em relação aos Algoritmos Genéticos puros é devido à utilização da técnica de Busca Tabu, que direciona a busca por melhores soluções na vizinhança de uma boa solução.

8.3. Tempo Computacional

As Figuras 14 a 17 mostram que o esforço computacional médio exigido pelo método híbrido *HBGATS pmx 05* para todas as classes (m,n) de problemas é, em muitos casos, superior aos demais métodos, mas não ultrapassa 30 segundos de tempo de computação para máquinas agrupadas e n = 100 tarefas, não sendo portanto excessivo.

Tendo em vista os resultados relatados neste trabalho, verifica-se que o método heurístico híbrido *HBGATS* com o operador de cruzamento PMX é muito eficiente para a solução de problemas de programação de operações em ambiente *Flow-Shop Permutacional*, garantindo uma boa solução em um pequeno intervalo de tempo computacional.

As pesquisas futuras poderão utilizar-se de diferentes operadores de cruzamento relatados na literatura, com o objetivo de obter melhores soluções para o problema de programação *Flow-Shop* Permutacional. Poderão, também, utilizar-se de operadores de cruzamento "inteligentes" (não-aleatórios), usando as características genéticas favoráveis (posições relativas das tarefas) dos cromossomos (seqüências) pais.

REFERÊNCIAS BIBLIOGRÁFICAS

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.

GAREY, M.R.; JOHNSON, D.S.; SETHI, R. (1976). The Complexity of Flow-Shop and Job-Shop 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.

JOHNSON, L.A., MONTGOMERY, D.C. (1974), *Operations Research in Production Planning, Scheduling and Inventory Control,* Wiley, New York

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). Adressing 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 $n/m/C_{max}$ Flow-shop 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 flow-shops. *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 Flow-Shop Sequencing. *Computers & Operations Research*, v.22, p.5-13.

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.

SUE, A. H. (1998). A hybrid heuristic for the uncapacitated hub location problem. *European Journal of Operational Research*, v.106, p.489-499.

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.

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.

ANEXO 1

Resultados dos Problemas do Experimento

(métodos pmxGA, ModFshop.TS5 e *HBGATS* *pmx 05* – operador de cruzamento PMX)

Os resultados estatísticos relativos aos métodos pmxGA, ModFshop.TS5 e *HBGATS pmx05* para máquinas **não-agrupadas** estão relatados nas Tabelas 23 a 25.

TABELA 23: Porcentagens de Sucesso para máquinas não-agrupadas (pmxGA, ModFshop.TS5 e *HBGATS pmx05*)

Algoritmo	Número de Máquinas	Número de tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	50,0	60,0	70,0	55,0	70,0	70,0	70,0	60,0	85,0
	7	15,0	5,0	15,0	0,0	10,0	25,0	30,0	40,0	40,0
	10	0,0	0,0	0,0	0,0	5,0	5,0	0,0	5,0	0,0
ModFshop.TS5	4	75,0	100,0	100,0	90,0	95,0	85,0	95,0	90,0	100,0
	7	65,0	55,0	50,0	75,0	65,0	70,0	60,0	65,0	85,0
	10	30,0	25,0	35,0	20,0	35,0	30,0	25,0	45,0	30,0
HBGATS pmx 05	4	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
	7	95,0	80,0	85,0	85,0	95,0	90,0	100,0	95,0	85,0
	10	75,0	85,0	90,0	100,0	90,0	90,0	95,0	95,0	100,0

TABELA 24: Melhorias Relativas (em porcentagem) para máquinas não-agrupadas (pmxGA, ModFshop.TS5 e *HBGATS pmx 05*)

Algoritmo	Número de Máquinas	Número de tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	0,14	0,13	0,11	0,02	0,01	0,00	0,01	0,01	0,00
	7	0,12	0,03	0,07	0,01	0,01	0,01	0,01	0,01	0,00
	10	0,16	0,09	0,01	0,01	0,00	0,00	0,00	0,00	0,02
ModFshop.TS5	4	0,51	0,31	0,22	0,14	0,11	0,04	0,08	0,08	0,02
	7	1,49	1,68	1,19	1,22	0,80	0,54	0,51	0,33	0,36
	10	2,00	2,73	2,22	1,65	1,48	1,14	1,18	0,96	0,77
HBGATS pmx 05	4	0,85	0,31	0,22	0,15	0,12	0,05	0,09	0,08	0,02
	7	1,94	2,19	1,50	1,32	0,93	0,57	0,60	0,43	0,36
	10	2,71	3,49	3,06	2,28	2,02	1,60	1,60	1,34	1,05

TABELA 25: Tempos Médios de Computação (em segundos) para máquinas não-agrupadas (pmxGA, ModFshop.TS5 e *HBGATS pmx 05*)

Algoritmo	Número de Máquinas	Número de tarefas								
		20	30	40	50	60	70	80	90	100
pmxGA	4	0,3	2,9	8,5	14,0	20,1	40,6	42,2	40,7	88,7
	7	0,2	1,9	4,7	3,7	18,3	10,3	4,6	20,5	18,8
	10	0,0	1,1	3,2	2,6	6,9	8,2	14,1	18,5	20,4
ModFshop.TS5	4	0,2	0,9	2,0	3,6	5,2	6,8	8,0	9,2	10,0
	7	0,4	2,1	4,7	7,7	10,4	6,7	7,6	9,0	10,2
	10	0,7	3,3	6,7	8,0	8,7	11,6	16,2	16,0	17,4
HBGATS pmx 05	4	0,3	1,6	3,5	5,6	8,2	9,7	13,3	13,9	19,4
	7	0,4	2,9	11,4	14,1	17,8	15,6	14,1	19,1	18,1
	10	0,7	3,4	9,7	16,6	23,2	27,8	34,4	38,8	43,7

Os resultados estatísticos relativos aos métodos pmxGA, ModFshop.TS5 e *HBGATS pmx 05* para **máquinas agrupadas** estão relatados nas Tabelas 26 a 28.

TABELA 26: Porcentagens de Sucesso para máquinas agrupadas

(pmxGA, ModFshop.TS5 e *HBGATS pmx 05*)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
PmxGA	21,7	21,7	28,3	18,3	28,3	33,3	33,3	35,0	41,7
ModFshop.TS5	56,7	60,0	61,7	61,7	65,0	61,7	60,0	66,7	71,7
HBGATS pmx 05	90,0	88,3	91,7	95,0	95,0	93,3	98,3	96,7	95,0

TABELA 27: Melhorias Relativas (em porcentagem) para máquinas agrupadas

(pmxGA, ModFshop.TS5 e *HBGATS pmx 05*)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
PmxGA	0,14	0,08	0,06	0,01	0,01	0,00	0,00	0,01	0,01
ModFshop.TS5	1,34	1,57	1,21	1,00	0,80	0,57	0,59	0,46	0,38
HBGATS pmx 05	1,83	1,99	1,59	1,25	1,02	0,74	0,76	0,62	0,48

TABELA 28: Tempos Médios de Computação (em segundos) para máquinas agrupadas

(pmxGA, ModFshop.TS5 e *HBGATS pmx 05*)

Algoritmo	Número de Tarefas								
	20	30	40	50	60	70	80	90	100
pmxGA	0,2	2,0	5,5	6,8	15,1	19,7	20,3	26,6	42,6
ModFshop.TS5	0,4	2,1	4,5	6,5	8,1	8,4	10,6	11,4	12,6
HBGATS pmx 05	0,5	2,6	8,2	12,1	16,4	17,7	20,6	23,9	27,1

ANEXO 2

Programas Computacionais

IMPLEMENTAÇÃO COMPUTACIONAL

VARIÁVEIS E CONSTANTES GLOBAIS

```
CONST
  TabuSize = 7 ;

TYPE

  JobPair = RECORD
        JobIndex, position : INTEGER;
        END;

  TabuList = ARRAY[1..TabuSize] OF JobPair;


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

   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      : ARRMN;
  VAR UBG    : ARRNN );

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

```
PROCEDURE DKij(
   VAR k  : INTEGER;
   VAR P  : ARRMN;
   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;                     (* DKij *)

BEGIN                       (* UBGij *)
   k := 1 ;
   DKij(k,P,DK);
   FOR i:=1 TO n DO
     FOR j:=1 TO n DO BEGIN
       UBG[i,j] := DK[i,j] ;
       IF UBG[i,j] < 0 THEN UBG[i,j] := 0
       END;

   IF m > 2 THEN  FOR k:=2 TO (m-1) DO
     BEGIN
       DKij(k,P,DK);
       FOR i:=1 TO n DO
         FOR j:=1 TO n DO
           IF i=j THEN UBG[i,j] := INF
           ELSE BEGIN
                UBG[i,j] := UBG[i,j] + DK[i,j];
                IF UBG[i,j] < 0 THEN UBG[i,j] := 0 ;
              END
     END
END;                        (* UBGij *)


FUNCTION MAKESPAN(
   VAR m,n  : INTEGER;
   VAR S    : ARRN;
   VAR P    : ARRMN) : INTEGER;

   VAR
     u,v : INTEGER;
     C   : ARRMN;

   BEGIN                    (* MAKESPAN *)
     MAKESPAN := 0 ;
     C[1,2] := P[1,S[2]] ;
     FOR v:=3 TO n DO
       C[1,v] := C[1,v-1] + P[1,S[v]] ;
     FOR u:=2 TO m DO
```

```
        BEGIN
          C[u,2] := C[u-1,2] + P[u,S[2]] ;
          FOR v:=3 TO n DO
            IF C[u-1,v] >= C[u,v-1]
              THEN C[u,v] := C[u-1,v] + P[u,S[v]]
            ELSE   C[u,v] := C[u,v-1] + P[u,S[v]]
        END;
      MAKESPAN := C[m,n]
    END;                        (* MAKESPAN *)


PROCEDURE SndTIME(
      VAR secondsTime  : REAL);

      VAR  h,min,snd,hund : WORD;

  BEGIN                         (* SndTIME *)

   h := 0 ; min := 0 ; snd := 0 ; hund := 0 ;
   GETTIME(h,min,snd,hund);
   secondsTime := 3600.0*h + 60.0*min + 1.0*snd + hund/100.0;

  END;                          (* SndTIME *)


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 NotTabu(VAR x, y : INTEGER) : BOOLEAN;

      VAR k : INTEGER;

        BEGIN                   (* NotTabu *)

         NotTabu := TRUE;
         FOR k := 1 TO TabuSize DO
           IF (TL[k].JobIndex = x) AND (TL[k].position = y) THEN
             NotTabu := FALSE;

        END;                    (* NotTabu *)
```

```
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 NbHybridTSsequences(
    VAR n    : INTEGER;
    VAR coef : REAL ) : INTEGER;

    VAR AUX  : REAL;

     BEGIN                    (* NbHybridTSsequences *)

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

     END;                     (* NbHybridTSsequences *)


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

  BEGIN                  { * NbSequencesGAandHybrid * }
    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;                { * NbSequencesGAandHybrid * }


PROCEDURE GetRandomNotTabuShiftNeighbour(
    VAR seq1, Nseq1, INseq1  : ARRN;
    VAR JOBsh, POSITIONh : INTEGER);

    VAR
     h, i, j, aux  : INTEGER;


    BEGIN              (* GetRandomNotTabuShiftNeighbour *)

     INseq1 := seq1;

     RANDOMIZE;

     REPEAT
      h := Random(n-1);
      h := h + 2;
      i := Random(n-1);
      i := i + 2;
```

```
    UNTIL (h <> i)  AND  (NotTabu(seq1[h],i) = TRUE) ;

    IF h < i THEN
     BEGIN
      aux := seq1[h];
      FOR j := h  TO (i-1) DO
       begin
         seq1[j] := seq1[j+1];
       end;
      seq1[i] := aux;
     END
    ELSE
     BEGIN
      aux := seq1[h];
      FOR j := h DOWNTO (i+1) DO
       begin
         seq1[j] := seq1[j-1];
       end;
      seq1[i] := aux;
     END;
    Nseq1 := seq1;
    JOBsh := seq1[i]; POSITIONh := h ;

  END;                (* GetRandomNotTabuShiftNeighbour *)


PROCEDURE GetBestTabuSearch5Neighbour(
    VAR seq2, BNseq2 : ARRN;
    VAR DBNseq2      : INTEGER;
    VAR TabuJ        : INTEGER;
    VAR TabuP        : INTEGER;
    VAR NbSequences  : INTEGER);

    VAR
      Nseq2, INseq2    : ARRN;
      DNseq2           : INTEGER;
      JOBsh, POSITIONh : INTEGER;
      NbNeighbours     : INTEGER;

    BEGIN               (* GetBestTabuSearch5Neighbour *)

      DBNseq2 := INF ;
      NbNeighbours := 0;

      REPEAT

       GetRandomNotTabuShiftNeighbour(
           seq2, Nseq2, INseq2, JOBsh, POSITIONh);
       seq2 := INseq2;
       DNseq2 := MAKESPAN(m,n,Nseq2,P);
       NbNeighbours := NbNeighbours + 1 ;

       IF DNseq2 < DBNseq2 THEN
         BEGIN
          BNseq2 := Nseq2 ; DBNseq2 := DNseq2 ;
```

```
          TabuJ := JOBsh; TabuP := POSITIONh ;
        END;

      UNTIL NbNeighbours = NbSequences;

    END.                (* GetBestTabuSearch5Neighbour *)


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

   VAR
    i, j, CutPosition  : INTEGER ;
    u, v, vaux         : INTEGER;
    NbCrossATTEMPT     : INTEGER;
    AUXs3, AUXs4       : ARRN;
    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;

      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;

      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 : ARRN);

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

  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 : ARRN;
      VAR s3, s4 : ARRN);

  VAR
    AuxD  : INTEGER;
    AuxS  : ARRN;
```

```
  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 TSforHybrid(
    VAR P       : ARRMN;
    VAR s       : ARRN;
    VAR BS      : ARRN;
    VAR BD      : INTEGER;
    VAR TL      : TabuList;
    VAR tsK     : INTEGER;
```

```
    VAR tsBS      : ARRN;
    VAR tsBD      : INTEGER);

VAR
  j          : INTEGER;
  TabuJob    : INTEGER;
  TabuPosition : INTEGER;
  DBNs       : INTEGER;
  BNs        : ARRN;

BEGIN                          (* TSforHybrid *)

   GetBestTabuSearch5Neighbour(s, BNs, DBNs,
                    TabuJob, TabuPosition, tsK ) ;

       tsBS := BNs ; tsBD := DBNs ;

       FOR j := TabuSize DOWNTO 2 DO
         TL[j] := TL[j-1] ;
       TL[1].JobIndex := TabuJob ;
       TL[1].position := TabuPosition ;

   IF DBNs < BD THEN
     BEGIN
       BS := BNs ;
       BD := DBNs ;
     END;

END;                           (* TSforHybrid *)
```

SOLUÇÃO INICIAL DO MÉTODO FSHOPH

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

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

  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;                          (* FITSP *)
```

ALGORITMO NEH

```
PROCEDURE NEH(
  VAR m,n        : INTEGER;
  VAR P          : ARRMN;
  VAR SEQUENCE   : ARRN;
  VAR D          : INTEGER);

  VAR
```

```
  i          : INTEGER;
  LPTSeq     : ARRN;


PROCEDURE LPTJobSort(VAR LPTSeq : ARRN);

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 *)
     before := 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 before > after;

     IF start < after THEN
      QUICKSORT(start, after);
     IF before < finish THEN
      QUICKSORT(before,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 : ARRN);

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


   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      : LONGINT;
  VAR SR         : REAL);

VAR
  K              : LONGINT;
```

```
  NbMut1, NbMut2, NbMut3     : LONGINT;
  Mut1, Mut2, Mut3           : INTEGER;
  seq1, seq2, seq3, seq4     : ARRN;
  Dseq1, Dseq2, Dseq3, Dseq4 : INTEGER;

  i, SameJobs                : INTEGER;
  SameParents                : BOOLEAN;


BEGIN                              (* onecutGA *)

   seq1 := CSeq[1]; seq2 := CSeq[2] ;
   NbSeq := 0 ;
   SR := 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;

   NbSequencesGAandHybrid( 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 ;
              NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
            END;
          END
```

```
        ELSE
        IF Dseq4 < BD  THEN
          BEGIN
            BS := seq4 ;
            BD := Dseq4 ;
            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;
        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;
        END;

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

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

    IF Dseq4 < BD  THEN
      BEGIN
         BS := seq4;
         BD := Dseq4;
      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 ;
            END;
           END
          ELSE
           IF Dseq4 < BD  THEN
            BEGIN
             BS := seq4 ;
             BD := Dseq4 ;
            END;

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

        IF Dseq1 < BD  THEN
          BEGIN
             BS := seq1;
             BD := Dseq1;
          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 ;
    END
  ELSE
    BEGIN
      SR := NbSeq / K ;
    END;

END;                        (* onecutGA *)
```

ALGORITMO GENÉTICO (OPERADOR PMX)

```
PROCEDURE pmxGA(
    VAR P           : ARRMN;
    VAR CSeq        : StartSEQS;
    VAR DCSeq1, DCSeq2   : INTEGER;
    VAR BS          : ARRN;
    VAR BD          : INTEGER;
    VAR NbSeq       : LONGINT;
    VAR SR          : REAL);

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

  i, SameJobs           : INTEGER;
  SameParents           : BOOLEAN;


BEGIN                           (* pmxGA *)

    seq1 := CSeq[1]; seq2 := CSeq[2] ;
    NbSeq := 0 ;
    SR := 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;

    NbSequencesGAandHybrid( 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 ;
           NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;


         END;
        END
        ELSE
         IF Dseq4 < BD  THEN
          BEGIN
            BS := seq4 ;
            BD := Dseq4 ;
            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;
        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 );
      Dseq4 := MAKESPAN( m, n, seq4, P );

      IF Dseq3 < BD  THEN
        BEGIN
           BS := seq3;
           BD := Dseq3;
        END;
```

```
      IF Dseq4 < BD  THEN
        BEGIN
          BS := seq4;
          BD := Dseq4;
        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 ;
          NbMut1 := NbSeq; NbMut2 := NbSeq; NbMut3 := NbSeq;
         END;
        END
        ELSE
        IF Dseq4 < BD  THEN
         BEGIN
          BS := seq4 ;
          BD := Dseq4 ;
          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;
        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;
      END;

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

    IF Dseq4 < BD  THEN
      BEGIN
         BS := seq4;
         BD := Dseq4;
      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 ;
         END;
       END
      ELSE
       IF Dseq4 < BD  THEN
        BEGIN
         BS := seq4 ;
         BD := Dseq4 ;
        END;

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

    IF Dseq1 < BD  THEN
      BEGIN
         BS := seq1;
         BD := Dseq1;
      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 ;
    END
  ELSE
    BEGIN
      SR := NbSeq / K ;
    END;

END;                          (* pmxGA *)
```

ALGORITMO ModFshopTS5

```
PROCEDURE ModFshopTS5(
    VAR P       : ARRMN;
    VAR InitSeq   : ARRN;
    VAR DInitSeq  : INTEGER;
    VAR BS        : ARRN;
    VAR BD        : INTEGER;
    VAR NbSeq     : LONGINT;
    VAR SR        : REAL);

VAR
  j          : INTEGER;
  nbmax      : INTEGER;
  Nbiter, BIter  : INTEGER;
  NbTS5Neighb   : INTEGER;
  TabuJob       : INTEGER;
  TabuPosition  : INTEGER;
  Ds, DBNs      : INTEGER;
  s, BNs        : ARRN;


BEGIN                         (* ModFshopTS5 *)

   FOR j := 1 TO TabuSize DO
     BEGIN
      TL[j].JobIndex := 0;
      TL[j].position := 0;
     END;

   NbSeq := 0 ;
   SR := 0.0 ;
```

```
s := InitSeq ;
Ds := DInitSeq ;

BS := s ; BD := Ds ;

NbTS5Neighb := NbTS5Neighbours(n) ;

nbmax := 0 ;

IF (m <= 20) AND (n <= 31) THEN
  nbmax := n-1 ;
IF (m < 15) AND (32 <= n) AND (n <= 50) THEN
  nbmax := n-1 ;
IF (m < 10) AND (51 <= n) AND (n <= 66) THEN
  nbmax := n-1 ;
IF (m < 7) AND (67 <= n) AND (n <= 102) THEN
  nbmax := 65 ;
IF (15 <= m) AND (m <= 20) AND (32 <= n) AND (n < 51) THEN
  nbmax := 30 ;
IF (10 <= m) AND (m <= 20) AND (51 <= n) AND (n < 67) THEN
  nbmax := 30 ;
IF (7 <= m) AND (m < 15) AND (67 <= n) AND (n <= 102) THEN
  nbmax := 30 ;
IF (15 <= m) AND (m <= 20) AND (67 <= n) AND (n <= 102) THEN
  nbmax := 10 ;

Nbiter := 0 ; BIter := 0 ;

WHILE (Nbiter - BIter) < nbmax DO
   BEGIN
    GetBestTabuSearch5Neighbour(s, BNs, DBNs,
                TabuJob, TabuPosition,
                NbTS5Neighb) ;

    NbSeq := NbSeq + NbTS5Neighb ;

    s := BNs ; Ds := DBNs ;
    Nbiter := Nbiter + 1 ;

    FOR j := TabuSize DOWNTO 2 DO
      TL[j] := TL[j-1] ;
    TL[1].JobIndex := TabuJob ;
    TL[1].position := TabuPosition ;

    IF DBNs < BD THEN
      BEGIN

        BS := BNs ;
        BD := DBNs ;
        BIter := Nbiter ;
      END;

   END; { WHILE }

SR := 1.0 ;
```

```
END;                    (* ModFshopTS5 *)
```

ALGORITMO HÍBRIDO HBGATS (OPERADOR DE 1 CORTE)

```
PROCEDURE Hybrid1XGAandModTS5(
    VAR P          : ARRMN;
    VAR CSeq       : StartSEQS;
    VAR DCSeq1, DCSeq2  : INTEGER;
    VAR BS         : ARRN;
    VAR BD         : INTEGER;
    VAR NbSeq      : LONGINT;
    VAR SR         : REAL);

VAR
  K                : LONGINT;
  seq1, seq2, seq3, seq4    : ARRN;
  Dseq1, Dseq2, Dseq3, Dseq4  : INTEGER;
  NewSeq1, NewSeq2          : ARRN;
  DNewSeq1, DNewSeq2        : INTEGER;
  tsK              : INTEGER;

  i, j, SameJobs          : INTEGER;
  SameParents             : BOOLEAN;


BEGIN                   (* Hybrid1XGAandModTS5 *)

   seq1 := CSeq[1]; seq2 := CSeq[2] ;

   FOR j := 1 TO TabuSize DO

     BEGIN
      TL[j].JobIndex := 0;
      TL[j].position := 0;
     END;

   tsK := 0 ; K := 0 ;
   NbSeq := 0 ;
   SR := 0.0 ;

   IF DCSeq1 <= DCSeq2  THEN
     BEGIN
        BS := CSeq[1] ;
        BD := DCSeq1 ;
     END
   ELSE
     BEGIN
        BS := CSeq[2] ;
        BD := DCSeq2 ;
     END;
   tsK := NbHybridTSsequences( n, Percent ) ;
   NbSequencesGAandHybrid( m, n, K );
```

```
    RANDOMIZE;

    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 );

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

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

TSforHybrid(P, seq1, BS, BD, TL, tsK, NewSeq1, DNewSeq1);
   NbSeq := NbSeq + tsK ;

TSforHybrid(P, seq2, BS, BD, TL, tsK, NewSeq2, DNewSeq2);
   NbSeq := NbSeq + tsK ;

        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 ;
    END
  ELSE
    BEGIN
      SR := NbSeq / K ;
    END;

 END;                          (* Hybrid1XGAandModTS5 *)
```

ALGORITMO HÍBRIDO HBGATS (OPERADOR PMX)

```
PROCEDURE HybridpmxGAandModTS5(
    VAR P          : ARRMN;
    VAR CSeq       : StartSEQS;
    VAR DCSeq1, DCSeq2  : INTEGER;
    VAR BS         : ARRN;
    VAR BD         : INTEGER;
    VAR NbSeq      : LONGINT;
    VAR SR         : REAL);

VAR
  K                : LONGINT;
  seq1, seq2, seq3, seq4    : ARRN;
  Dseq1, Dseq2, Dseq3, Dseq4  : INTEGER;
  NewSeq1, NewSeq2          : ARRN;
  DNewSeq1, DNewSeq2        : INTEGER;
  tsK              : INTEGER;

  i, j, SameJobs         : INTEGER;
  SameParents            : BOOLEAN;


BEGIN                          (* HybridpmxGAandModTS5 *)

   seq1 := CSeq[1]; seq2 := CSeq[2] ;

   FOR j := 1 TO TabuSize DO
     BEGIN
      TL[j].JobIndex := 0;
      TL[j].position := 0;
     END;

   tsK := 0 ; K := 0 ;
   NbSeq := 0 ;
   SR := 0.0 ;

   IF DCSeq1 <= DCSeq2  THEN
     BEGIN
        BS := CSeq[1] ;
        BD := DCSeq1 ;
     END
   ELSE
     BEGIN
        BS := CSeq[2] ;
        BD := DCSeq2 ;
     END;
   tsK := NbHybridTSsequences( n, Percent ) ;
   NbSequencesGAandHybrid( m, n, K );

   RANDOMIZE;

   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 ;
         END;

TSforHybrid(P, seq1, BS, BD, TL, tsK, NewSeq1, DNewSeq1);
   NbSeq := NbSeq + tsK ;

TSforHybrid(P, seq2, BS, BD, TL, tsK, NewSeq2, DNewSeq2);
   NbSeq := NbSeq + tsK ;

        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 ;
    END
  ELSE
    BEGIN
      SR := NbSeq / K ;
    END;

END;                          (* HybridpmxGAandModTS5 *)
```