ALGORITMO METAHEURISTICO PARA BUSCA DO GARGALO FLUTUANTE EM FLOW SHOP PERMUTACIONAL COM TEMPOS DE SETUP ASSIMÉTRICOS E DEPENDENTES DA SEQÜÊNCIA

Alexandre Damas de Barros

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

ORIENTADOR: Prof. Titular João Vitor Moccellin

São Carlos / SP

2002

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

B277a

Barros, Alexandre Damas de
Algoritmo metaheuristico para busca do gargalo flutuante em flow shop permutacional com tempos de setup assimétricos e dependentes da sequência / Alexandre Damas de Barros. -- São Carlos, 2002.

Dissertação (Mestrado) -- Escola de Engenharia de São Carlos-Universidade de São Paulo, 2002.
Área: Engenharia de Produção.
Orientador: Prof. Titular João Vitor Moccellin.

1. Gargalos. 2. Algoritmo metaheuristico. 3. Setup dependente da sequência. 4. Scheduling. I. Título.

FOLHA DE JULGAMENTO

Candidato: Tecnólogo **ALEXANDRE DAMAS DE BARROS**

Dissertação defendida e julgada em 20-12-2002 perante a Comissão Julgadora:

Prof. Tit. **JOÃO VITOR MOCCELLIN (Orientador)** — APROVADO
(Escola de Engenharia de São Carlos/USP)

Prof. Dr. **MARCELO SEIDO NAGANO** — APROVADO
(Faculdade de Economia e Administração/Campus de Ribeirão Preto/USP)

Prof. Dr. **FLÁVIO CÉSAR FARIA FERNANDES** — APROVADO
(Universidade Federal de São Carlos/UFSCar)

Prof. Doutor **EDMUNDO ESCRIVÃO FILHO**
Coordenador do Programa de Pós-Graduação
em Engenharia de Produção

Profa. Assoc. **MARIA DO CARMO CALIJURI**
Presidente da Comissão de Pós-Graduação da EESC

A minha mãe. Saudades.

Ao Professor João Vitor Moccellin, pela excelente orientação e principalmente por acreditar e confiar no meu trabalho.

A todos os colegas, funcionários e professores do Departamento de Engenharia de Produção da EESC/USP pela excelente estrutura mantida e colaboração.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

BARROS, A.D. (2002). Algoritmo Metaheuristico para Busca do Gargalo Flutuante em Flow Shop Permutacional com Tempos de Setup Assimétricos e Dependentes da Seqüência. São Carlos, 2002. 112p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

Trata do desenvolvimento de um algoritmo metaheurístico para *flow shop* permutacional com tempos de *setup* assimétricos e dependentes da seqüência (***ASDST***), com foco na principal restrição à saída de um sistema de produção definido: o gargalo. O algoritmo desenvolvido, chamado "Algoritmo Metaheuristico de Busca do Gargalo Flutuante em *Simulated Annealing* (***BGaFSA***)" tem como critério de desempenho a otimização do *makespan* ($C_{max.}$) e analisa uma possível mudança do gargalo no sistema decorrente da otimização local do gargalo inicialmente identificado. A partir do desenvolvimento de um *software*, analisou-se o comparativo com outro algoritmo bem referenciado na literatura e também o comportamento individual do *BGaFSA*. A experimentação computacional demonstra que a simples otimização local do gargalo pode não ser suficiente para a obtenção do melhor resultado global, e que a flutuação do gargalo ocorre e deve ser considerada para a obtenção de uma solução ótima global ao problema. Isso contraria parte das pesquisas em ***ASDST*** com gargalos que considera a otimização local do gargalo como o melhor caminho para a otimização do sistema completo.

Palavras-chave: ***gargalos, algoritmo metaheuristico, setup dependente da seqüência, scheduling.***

ABSTRACT

BARROS, A.D. (2002). Metaheuristic Algorithm for Search of the Flotation Bottleneck in Permutational Flow Shop with Asymmetric Sequence Dependent Setup Times. São Carlos, 2002. 112p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

It treats of the development of a metaheuristic algorithm for permutational flow shop with asymmetric sequence dependent setup times (*ASDST*), with focus in the main restriction to the output of a defined production system: the bottleneck. The developed algorithm, called "Metaheuristic Algorithm of Search of the Flotation Bottleneck in Simulated Annealing (*BGaFSA*)", it has as acting criterion the optimization of the makespan (*Cmax.*) and it analyzes a possible change of the bottleneck in the system due to the local optimization of the bottleneck initially identified. Starting from the development of a software, the comparative was analyzed with other algorithm well referenced in the literature and also the individual behavior of *BGaFSA*. The computational experimentation demonstrates that the simple local optimization of the bottleneck cannot be enough for the obtaining of the best global result, and that the flotation of the bottleneck happens and it should be consider for the obtaining of a global great solution to the problem. That contradicts part of the researches in *ASDST* with bottleneck that it considers the local optimization of the bottleneck as the best road for the optimization of the complete system.

Keywords: ***bottleneck, metaheuristic algorithm, sequence dependent setup, scheduling.***

CAPÍTULO 1

PLANEJAMENTO E PROGRAMAÇÃO DE PRODUÇÃO

1.1. Planejamento e Controle

Para SLACK et. al. (1996), o planejamento e controle preocupam-se em gerenciar as atividades da operação produtiva de modo a satisfazer a demanda dos consumidores. Qualquer operação produtiva necessita de planejamento, desde as consideradas mais simples quanto às mais complexas. A atividade de planejamento e controle pode ser considerada como segue:

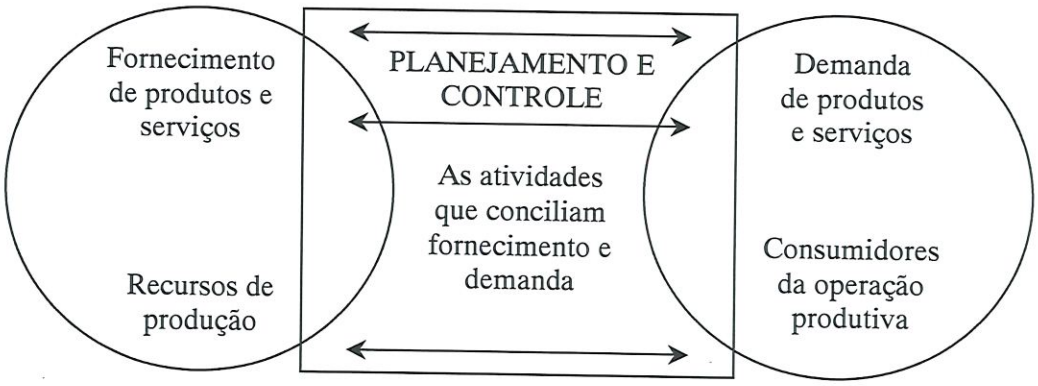


FIGURA 1: a função de planejamento e controle em um sistema de produção definido (SLACK et al., 1996).

Planejamento e controle cuidam da interligação entre os componentes do processo produtivo e suas interligações com o atendimento à demanda. Agregam desde o fluxo de matéria-prima e insumos à produção quanto à programação da

produção e distribuição física dos produtos e são, sobretudo, responsáveis por gerenciar o fluxo de produtos e informações desde a previsão/geração da demanda até o atendimento dessa demanda, garantindo que a produção ocorra eficazmente e garanta produtos e/ou serviços na quantidade adequada, no momento adequado e no nível de qualidade adequado e/ou esperado.

Outra função atrelada ao planejamento e controle é quanto ao gerenciamento de custos na empresa. Administrar a dicotomia flexibilidade de produção com baixos custos de inventário (insumos e produtos acabados) e seus reflexos no atendimento à demanda é uma das funções mais difíceis em um ambiente fabril, dadas as diferenças de critérios de desempenho atribuídas a cada área, onde o atendimento a um objetivo de desempenho muitas vezes afeta o atendimento a outro objetivo do sistema. Estabelecer um sistema de prioridades adequado aos objetivos da empresa é fundamental à correta estruturação do planejamento e controle em uma empresa.

1.2. Planejamento e Programação de Produção

Para SLACK et al. (1996), a atividade de programação é uma das tarefas mais complexas do gerenciamento de produção em função de:

- os programadores têm que lidar com diferentes tipos de recurso simultaneamente;
- as máquinas e equipamentos têm diferentes capacidades e condições;
- a programação de n tarefas possibilita $n!$ programações possíveis;
- para tarefas manuais há a influência direta da habilidade individual de cada operador, diminuindo a previsibilidade de conclusão e/ou início das tarefas ou operações no sistema;
- conforme o tipo de processo, há a necessidade de concatenar a disponibilidade de dezenas de itens diferentes simultaneamente, o que pode levar a reprogramações e afetar negativamente o critério de desempenho adotado;

Conseguir concatenar todos esses itens de forma a atender aos critérios de desempenho estabelecidos é a principal função do planejamento e programação de produção. A grande dificuldade da programação de operações é quanto às variações dinâmicas que afetam a capacidade produtiva. Por esses fatores ocorrerem de forma dinâmica tendem a serem imprevisíveis e levam a necessidades de reprogramações das tarefas. Alguns desses fatores são:

- quebra de equipamentos (indisponibilidade de equipamentos em conseqüência de falhas);
- absenteísmo (ausências não previstas de mão de obra);
- não-conformidades no processo (problemas de qualidade com o processo que afetam o produto e reduzem a disponibilidade do produto ao cliente ou o fluxo do processo).

Devido ao excessivo número de variáveis que integram e afetam a eficácia da tarefa de programação de produção, alguns fatores são desprezados quando o *set* definido de tarefas a serem executadas forem sequeciadas. Geralmente esses fatores são absorvidos por uma folga entre capacidade bruta de produção e a demanda real, e variam em função do tipo de processo e modelo administrativo.

A função da programação de produção é planejar as atividades, reportar os resultados operacionais e revisar os planos requeridos para atingir os resultados desejados (FOGARTY, BLACKSTONE JR & HOFFMANN, 1983). Pode também ser executada de diferentes formas, dependendo do tipo de processo, do tipo de produto, da medida de desempenho adotada à fábrica e também da filosofia administrativa empregada, como por exemplo:

- em uma fábrica que foque a máxima utilização de gargalos de produção, a programação será definida priorizando esse(s) recurso(s). Isso levará a reprogramações e adequações dos outros recursos do sistema;
- sendo o produto de alto valor agregado, a programação priorizará os estoques, mantendo o custo de inventário no nível mínimo;
- caso um produto seja sazonal, pode-se produzir por demanda real ou por demanda planejada (*make to order* ou *make to stock*). Isso afeta a

programação quanto a insumos, utilização de recursos e nível de utilização de mão de obra;

- sendo o atendimento ao cliente a medida de desempenho adotada, a programação será executada priorizando os prazos de entrega, o que pode muitas vezes "piorar" outras medidas, como por exemplo produtividade. Outro fator que pode ser "piorado" por esse critério é o de custos de operação, como por exemplo, o aumento dos custos com *setup* de máquinas em função da diminuição do tamanho dos lotes de produção;
- pode variar conforme o tipo de ambiente de produção (fluxo contínuo ou repetitivo, fluxo por lotes, células de manufatura e processos intermitentes (FOGARTY, BLACKSTONE JR & HOFFMANN, 1983));
- em função do arranjo físico (*lay-out*), celular ou departamentalizado, a tarefa de programação poderá ser diferenciada e adaptada a cada um.

Por fim, podemos afirmar que a tarefa de programação varia de empresa para empresa, varia ainda conforme o critério de desempenho e/ou priorização considerado e quanto à situação do momento (política, recursos financeiros, liquidez da empresa e outros). É, portanto, uma tarefa dinâmica e que precisa ser revista periodicamente para que possa refletir eficazmente à necessidade a que se destina.

1.3. Administração da Demanda e Gestão de Estoques

Um ponto importante na estratégia de uma empresa é a política de atendimento à demanda. Essa política pode variar conforme a situação político-econômica do país, a estabilidade da empresa no mercado, a situação financeira da empresa, o perfil dos clientes e outros. Em Economia diz-se que a demanda é gerada pelo mercado. Portanto administra-se uma informação gerada pela demanda, representada pela variação no consumo, e somente em raros casos a demanda é gerada pela empresa, como em casos de monopólio ou setores cartelizados. Normalmente a empresa somente influencia a demanda, com promoções, propagandas e outros. A demanda é administrada para dentro da empresa na forma de programação ao fator produtivo e conseqüentemente às áreas relacionadas, como

o gerenciamento de estoques. Para uma empresa é de fundamental importância a programação do fator produtivo, mais conhecido por *forecast,* e suas forma e controle. Para PLOSSL (1985), há muitos conflitos na geração da programação, pois determinada área da empresa vai privilegiar o embarque dos produtos, enquanto outra priorizará a facilidade de produção, uma terceira enfatizará a administação dos níveis de estoque e assim por diante. Essas diferenças de objetivos entre áreas pode levar a uma divergência em suas formas de atuação, pois cada área privilegiará o critério de desempenho a ela adotado e isso poderá levar a empresa a situações totalmente imprevistas. Vem daí a necessidade de modelos de objetivos coerentes à empresa como um todo, mesmo que isso signifique perder em algum setor.

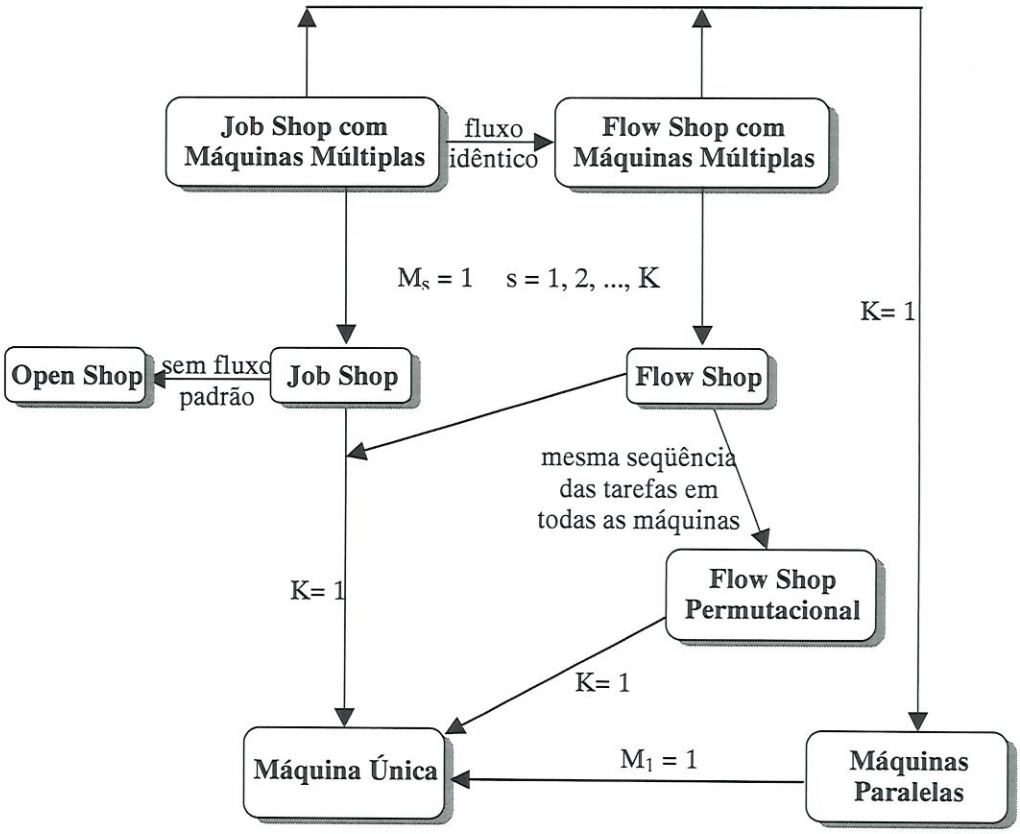
A situação mais crítica dentro da administração da demanda é a ocorrência da demanda imprevista e/ou com necessidade de atendimento menor que o *lead-time* de fabricação. Nesses casos o atendimento à demanda fica comprometido pela própria incapacidade quanto ao prazo de atendimento e pode levar às conseqüências comuns ao não atendimento à demanda, como perda do pedido e/ou do cliente. Em muitos casos, uma forma de absorção dessas variações é a existência de estoques, porém isso acarreta acréscimos nos custos ao produto final, já que estoques são recursos financeiros parados sobre os quais recaem custos como perda de oportunidade em função da falta de liquidez, custos financeiros desse recurso e outros (de acordo com ARAÚJO (1971), 75% do capital de uma empresa muitas vezes estão representados por aquisições de matérias-primas, equipamentos, máquinas, acessórios e outros itens). Em algumas situações o não atendimento à demanda pode ter multas decorrentes de obrigações contratuais legais, custos para informar ao cliente pedidos atrasados e a própria perda do cliente. Uma forma de entender e especificar esse impacto é entender e ter de forma clara os custos envolvidos com o atendimento à demanda imprevista e seu reflexo dentro da empresa. Essa análise passa por vários fatores, como valor agregado dos produtos, oferta e demanda do segmento no mercado, saúde financeira da empresa, política de atendimento à demanda e outras, variando conforme o tipo da empresa e principalmente conforme sua filosofia administrativa. Essa análise é fundamental para suportar o estabelecimento da política de atendimento à demanda de uma empresa.

1.4. Problemas de Programação de Operações em Máquinas

Para cada um dos vários ambientes possíveis em um sistema fabril um modelo de programação é necessário, pois em função dos ambientes serem muito específicos, a aplicação de um conceito de programação em outro ambiente pode ter resultados indesejados. A cada ambiente de programação podemos ligar um problema de programação específico, uma adaptação de um modelo existente ou ainda um modelo hibrido. Para MACCARTHY & LIU (1993), os problemas de programação podem ser definidos como segue:

- *JOB SHOP:* cada tarefa tem seu fluxo individual ou rota específica através das máquinas que devem ser utilizadas;
- *FLOW SHOP:* cada tarefa tem um fluxo idêntico às outras;
- *OPEN SHOP:* não há fluxo específico para nenhuma tarefa;
- *FLOW SHOP PERMUTACIONAL:* um *flow shop* onde a ordem de processamento das tarefas em todas as máquinas é estritamente a mesma;
- *MÁQUINA ÚNICA:* somente uma máquina é considerada.
- *MÁQUINAS PARALELAS:* k máquinas idênticas em um estágio simples de processo. Cada tarefa necessita uma, e somente uma destas máquinas;
- *JOB SHOP COM MÁQUINAS MÚLTIPLAS:* um *job shop* em que há k_i máquinas idênticas em cada estágio $(i= 1,2,....,m)$ e qualquer tarefa neste estágio necessita ser processada em uma, e somente em uma destas máquinas.
- *FLOW SHOP COM MÁQUINAS MÚLTIPLAS:* as tarefas são geralmente processadas em múltiplos estágios e para cada estágio há máquinas paralelas idênticas, variando a quantidade por estágio. Cada operação de cada tarefa é processada em uma, e somente em uma destas máquinas.

A figura 2 apresenta, esquematicamente, os problemas de programação existentes:



K = número de estágios de produção;
M_s = número de máquinas do estágio s.

FIGURA 2: relacionamento entre os ambientes de máquinas em fábrica (adaptado de MACCARTHY & LIU, 1993).

Para HAX & CANDEA (1984), o problema de programação de tarefas pode ainda ser classificado em estático e dinâmico. No caso estático, todas as tarefas a serem processadas na fábrica chegam simultaneamente, no mesmo tempo T (normalmente atribuído como zero). Essa situação geralmente ocorre em tarefas processadas por lotes, onde o lote seguinte é programado quando ainda há um lote sendo processado. Para o caso dinâmico, a chegada de tarefas à fábrica é intermitente, ou seja, chegam à fábrica em tempos diferentes. Essa chegada pode ser

planejada ou variar de acordo com algum processo probabilístico. O problema de programação pode ainda ser classificado com determinístico e probabilístico, conforme segue:

- *DETERMINÍSTICO:* para todas as tarefas, os atributos relativos à fábrica são conhecidos e fixados com precisão, ou seja, as tarefas ou são disponibilizadas no tempo zero ou deixam a fábrica em tempos futuros conhecidos e os tempos de processamento são conhecidos quando as tarefas chegam à fábrica;

- *PROBABILÍSTICO:* os tempos de conclusão das tarefas e/ou tempos de processamento são variáveis randômicas descritas por distribuições de probabilidade. Começa com um sistema de fila em que uma tarefa pode gerar uma programação curta. As atribuições típicas ao modelo são que as operações têm tempos de processamento exponencialmente distribuídos e os tempos obtidos dessa distribuição são disponibilizados aos procedimentos de programação.

Os problemas de programação também podem variar conforme os critérios de desempenho atribuídos. Para BAKER (1974), três tipos de decisões prevalecem em programação e indicam medidas de desempenho comumente usadas:

- *UTILIZAÇÃO EFICIENTE DE RECURSOS:* tempo de conclusão máxima de tarefas, ou Makespan, indicado por $C_{max.}$;

- *RESPOSTA RÁPIDA À DEMANDA:* tempo de conclusão médio $\overline{C}$, tempo de fluxo médio $\overline{F}$ ou tempo de espera médio $\overline{W}$;

- *TÉRMINO CONFORME TEMPO FINAL INDICADO:* atraso médio $\overline{T}$, atraso máximo $T_{max.}$ e o número de tarefas em atraso N_T.

As abordagens dos problemas de programação quanto à variação de chagada das tarefas à fábrica (estático e dinâmico), quanto à natureza dos tempos (determinístico e probabilístico) e quanto aos critérios de desempenho combinam-se com os ambientes de programação de fábrica criando novas abordagens aos problemas.

1.5. Objetivos e Estrutura do Trabalho

Esta pesquisa visa explorar o problema de *flow shop* com tempos de *setup* assimétricos e dependentes da seqüência com gargalo de produção e propor um método metaheurístico para solucionar o problema de minimização da duração total da programação (*makespan*) nesse ambiente, abordando o impacto da aplicação do método sobre os custos em empresas.

A motivação para proposição desta pesquisa baseia-se na dificuldade de métodos eficazes e aplicáveis para a otimização de sistemas de produção com tempos de *setup* assimétricos e dependentes da seqüência. Esse ambiente de produção é comum em industrias gráficas, químicas e sistemas com alto *mix* de produtos e baixos volumes de produção, sendo que para esses ambientes a criação de ferramentas aplicáveis é fundamental para maximizar a utilização dos recursos disponíveis. Aliado à disponibilidade de recursos tecnológicos atualmente existentes nas empresas, em especial ao chão de fábrica, um sistema eficaz para programação de tarefas torna-se de muita valia no dia a dia, onde muitas decisões são tomadas somente com base na experiência, muitas vezes desconsiderando-se os processos de otimização.

Todo o processo descrito tem como foco a principal restrição à saída do sistema, chamado gargalo, ou seja, toda a otimização será efetuada sobre o recurso gargalo e expandido para os demais recursos integrantes desse sistema. A diferença dos métodos tradicionais para resolução do problema apresentado reside em que o algoritmo proposto executa uma busca para identificar uma possível mudança no gargalo inicialmente identificado.

O trabalho foi estruturado como segue:

CAPÍTULO 1: aborda os aspectos básicos do planejamento e controle de produção e trata da função de custo atrelada à variação de *lead-time* de produção. Conclui apresentando os modelos de Problemas de Programação de Operações em Máquinas e os critérios de desempenho comumente abordados;

CAPÍTULO 2: define restrições e aborda suas influências em um sistema produtivo e seus impactos no processo como um todo;

CAPÍTULO 3: caracteriza e contextualiza o problema de programação de operações *flow shop* permutacional tradicional e com tempos de *setup* assimétricos e dependentes da seqüência *(ASDST)*. Discorre sobre as pesquisas desenvolvidas na área;

CAPÍTULO 4: apresenta o método metaheurístico *Simulated Annealing*, sua origem e principio de funcionamento e os parâmetros agregados à sua aplicação;

CAPÍTULO 5: apresenta o algoritmo proposto, seu desenvolvimento, estrutura e aplicação. Conclui com a seqüência lógica do algoritmo proposto.

CAPÍTULO 6: apresenta a estrutura do *software* desenvolvido e os resultados obtidos com a resolução dos problemas gerados. Trata da experimentação computacional e análise dos resultados obtidos do algoritmo proposto e sua comparação com outro algoritmo bem referenciado na literatura.

CAPÍTULO 7: analisa a aplicação e funcionamento do algoritmo proposto quanto a seus aspectos característicos;

CAPÍTULO 8: conclui o trabalho abordando os aspectos relevantes encontrados e apresenta propostas para pesquisas futuras.

CAPÍTULO 2

INFLUÊNCIAS DAS RESTRIÇÕES NA EFICIÊNCIA DO SISTEMA PRODUTIVO

2.1. Definição de Restrições ao Sistema Produtivo

Em um sistema produtivo definido, restrições são todos os processos, máquinas, meios ou até comportamentos que limitam o sistema, ou seja, limitam o máximo de sua performance. Segundo STEIN (1997), as restrições pode ser definidas como segue:

- *COMPORTAMENTAL:* quando o comportamento está em conflito com a realidade e resulta em um impacto negativo nas medidas globais da companhia, isso é dito como uma restrição comportamental. É diretamente afetado pelo treinamento, educação, sistemas de medição, experiências, atitudes e disposição mental das pessoas envolvidas;

- *GERENCIAIS:* políticas gerenciais pobres com freqüência restringem a habilidade para maximizar a utilização dos recursos físicos e suas estratégias de utilização. Como exemplo, o uso "pobre" de recursos dos sistema na tentativa de maximizar o lucro com base no critério de margem de lucro por produto;

- *CAPACIDADE:* uma restrição de capacidade existe toda vez em que uma demanda local em um recurso excede a capacidade disponível. Restrições de capacidade podem incluir máquinas, pessoas e podem restringir a saída do sistema;

- *MERCADO:* considerando que o mercado dita as regras quanto à aceitação de produtos, preços, *lead-time*, quantidade e qualidade de produtos, essa pode ser a principal restrição de todas. Quando a demanda de mercado é menor que a capacidade de recursos da companhia, é dito que há uma restrição de mercado;

- *LOGISTICA:* quaisquer problemas que ocorram originados dos sistemas de planejamento e controle da companhia são ditos como restrições logísticas. Como exemplo a falta de determinado componente que leve a uma perda de sincronia entre a necessidade de mercado e a produção de produtos.

Para o presente trabalho considera-se somente a restrição de capacidade por estar mais próxima à aplicação de um algoritmo de seqüenciamento de tarefas no chão-de-fábrica. No entanto todas as restrições estão intimamente ligadas, e uma afeta a outra em maior ou menor intensidade. Por essa abordagem, serão considerados os dois principais fatores considerados pelo algoritmo proposto dentro da restrição de capacidade que são o gargalo de produção e os tempos de *setup*.

2.2. Definição de Gargalos e sua Administração

Segundo GOLDRATT & FOX (1997), gargalos são operações que representam restrições à saída (ou *output)* do sistema de produção. Um gargalo pode ser definido por uma máquina, um processo ou outro recurso que seja identificado como limitante do *output* do sistema e o conceito existe com o objetivo de aumentar o ganho, não para reduzir as despesas operacionais.

Pelo exposto e considerando que o sistema esteja trabalhando no máximo de sua capacidade instalada, a administração do(s) gargalo(s) do sistema é crucial para o atendimento à demanda, pois significa manter a operação gargalo do sistema de produção transformando material pelo maior tempo disponível possível possibilitando um fluxo máximo e contínuo de produtos acabados no final do sistema de produção. Adaptando para uma linguagem mais próxima ao chão-de-fábrica, isso pode ser entendido também como reduzir ao máximo os tempos de espera entre

tarefas, os tempos ociosos, os tempos de *setup* e as indisponibilidades de equipamentos (manutenções corretivas, ajustes e outros). Para aumentar a eficiência de um sistema produtivo, o(s) gargalo(s) deve(m) ser administrado(s) em separado do restante do sistema e com todo o foco e cuidados possíveis e necessários sobre ele. Isso se justifica por como ser ele a restrição à saída do sistema, qualquer perda nesse recurso significa uma perda direta estendida a todo o sistema, posto que uma perda em outro recurso pode ser recuperada e a perda então diluída. Para GOLDRATT & FOX (1997), a administração do gargalo deve ser como segue:

- não pára em momento algum (refeições, intervalos, etc.);
- só produz peças boas (se necessário controlar as peças imediatamente antes do gargalo);
- foco total nessa operação (monitoramento de causas de paradas, manutenção preventiva/preditiva, flexibilidade de operação, etc.);
- proteger o gargalo com estoques para evitar que as flutuações e/ou problemas com as operações não-gargalo anteriores a ele afetem sua utilização constante;
- aumentar a capacidade do gargalo através de:
  - Retirar/tranferir operações para outros postos;
  - Duplicar (se necessário);
  - Eliminar/reduzir operações que não agregam valor;
  - Otimizar *setups*.

Com a aplicação dessas medidas ao recurso gargalo garante-se que ele não pare, mantendo sempre o fluxo máximo de saída de produtos do sistema. Por outro lado, a administração dos recursos não-gargalo deve ter como base o próprio recurso gargalo, cuidando tanto para não aumentar o estoque em processo quanto para que o recurso gargalo não pare por falta de peças.

2.3. Impacto da Redução dos Tempos de *Setup* em um Sistema Produtivo

2.3.1. Definição de Tempos de *Setup* entre Tarefas

Uma tarefa é composta por várias operações executadas conforme uma seqüência e com recursos estabelecidos, onde sua definição pode ser dada por um *set* de operações de transformação, inspeção ou movimentação que levam a uma mudança de *status* do produto, ou seja, levam a uma mudança de disponibilidade do produto ao processo seguinte. O tempo de execução de uma operação pode ser dividido como segue:

- tempo de preparação ou tempo de *setup* (tempo de preparação do recurso para a transformação da peça);
- tempo de processamento (ou transformação da peça);
- fila (tempo de espera por um recurso de processo);
- espera (tempo de espera por outra peça e/ou componente).

O tempo de preparação ou tempo de *setup* é o tempo despendido com a preparação de um recurso para a transformação/mudança de *status* de um produto. Pode ser uma troca de ferramentas, uma troca de dispositivos ou ferramental ou outra alteração no equipamento ou posto de trabalho necessária à execução da modificação no produto na operação seguinte. Conceitualmente o tempo de *setup* abrange desde o final de uma operação até o início da transformação do produto na operação seguinte.

Para FLYNN (1987) o tempo requerido para o *setup* é uma função direta do grau de similaridade entre duas operações que são processadas em uma mesma máquina em seqüência. Portanto se duas operações a serem processadas em seqüência são idênticas, o tempo requerido para o *setup* será relativamente pequeno. No entanto se forem completamente diferentes, o tempo será proporcionalmente maior. KIM & BOBROWSKI (1994) definem a importância do tempo de *setup* por:

- o tempo de *setup* constitui uma parte do tempo de fluxo que afeta diretamente a taxa de saída de um sistema de produção;
- o custo da unidade do tempo de *setup* usualmente é mais alto que o custo unitário do tempo de processamento, desde o custo de *setup* incluir perda de produtividade devido ao tempo de máquina parada até o custo de *setup* em si, isto é, o custo da mão de obra de um técnico de *setup* e
- o *setup* pode, com freqüência, ser feito somente por um técnico qualificado, já que o *setup* é, em muitos casos, um trabalho complicado requerendo um alto nível de especialização, e esse pessoal qualificado é um recurso limitado e nem sempre disponível.

LOCKETT & MUHLEMANN (1972) definiram de forma bastante clara o *setup* entre operações e sua ligação com a taxa de saída do sistema: "Uma tarefa consiste de um componente de engenharia requerendo várias operações, por exemplo, furação, torneamento, e pode usar em cada estação uma das ferramentas necessárias. As operações diferem consideravelmente em suas ferramentas requeridas e podem requerer ferramentas não disponíveis no magazine. Se isso ocorrer, as ferramentas em questão têm que ser trocadas com um considerável custo de mão de obra e tempo, e então reduzindo a taxa de saída do sistema. A máquina em questão está trabalhando na capacidade e o acúmulo de trabalho é crescente, e então um método de programação é requerido para incrementar a taxa de saída do sistema, por exemplo, minimizando os tempos de *setup*. Então, o objetivo do problema pode ser minimizar o número de troca de ferramentas". Essa abordagem é uma forma bastante simplificada e direta do problema do *setup*, completando as afirmações de FLYNN (1997) e KIM & BOBROWSKI (1994) quanto à similaridade entre as tarefas e custos de *setup*.

2.3.2. Impacto da Redução dos Tempos de *Setup* na Performance do Sistema

O tempo de *setup* necessário à execução de uma tarefa pode ser enxergado como uma função de custo ao processo, pois sua variação arrastará variações de

necessidade de pessoal para sua execução e também de tempo disponível de máquina. Uma redução em tempo de *setup* afeta diretamente o *lead-time* de processamento das tarefas com impacto direto nos estoque em processo, de insumos e final e todos os custos relativos a eles, como custos de obsolescência, de administação e financeiro (capital imobilizado e liquidez) . Com a variação no *lead-time* de produção, temos a variação direta na capacidade de atendimento à demanda na velocidade de sua ocorrência, ou seja, quanto menor for o *lead-time* de produção, maior será a possibilidade de atendimento à demanda em função da velocidade de entrega.

BLACK (1983) definiu a necessidade de redução dos tempos de *setup* como ligada diretamente à necessidade de lotes de produção menores, incluindo ainda:

- a proliferação no número de produtos resultando em necessidade de lotes de produção menores e impactando diretamente no custo de *setup*, o qual é enxergado como uma parcela do custo total de produção;
- a necessidade de *lead times* de produção menores para reduzir estoques e permitir respostas rápidas a mudanças na demanda;
- a variedade crescente de materiais diversos para garantir a flexibilidade e diversidade no processo, afetando o volume de estoques;
- a necessidade de aumentar a produtividade e reduzir custos.

YANG & DEANE (1993) desenvolveram trabalhos considerando reduções em tempos de *setup* em sistemas celulares, demonstrando que a redução nos tempos de *setup* pode reduzir o tempo médio e a variação no tempo de fluxo das tarefas e também a redução no tamanho do lote ótimo de produção. A figura 3 apresenta, mais especificamente, que a redução no tempo de fluxo de tarefas resulta em incrementos na velocidade de entrega, no nível de estoque em processo e em respostas rápidas a necessidades de mercado.

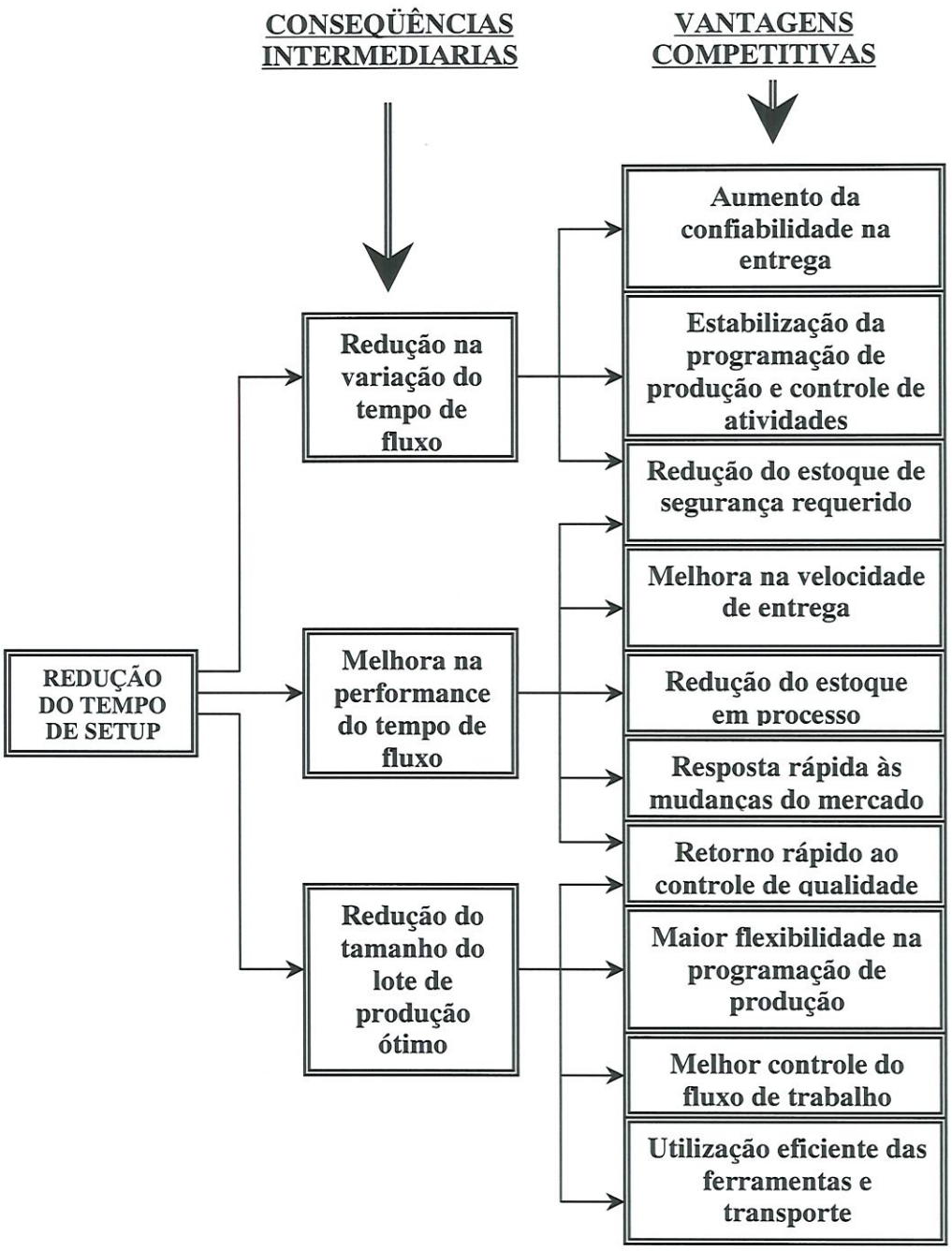


FIGURA 3: redução de tempos de *setup* e vantagens competitivas na manufatura (YANG & DEANE, 1993).

KARMARKAR (1987) abordou o problema de redução dos tempos de *setup* e seu impacto no *lead time* de produção ligado aos problemas de programação

de tarefas e atendimento à demanda. Em resumo, as considerações feitas seguem abaixo:

- longos *lead-times* forçam a programação final a ser congelada por longos períodos, enquanto cresce a possibilidade de mudanças na programação;
- o aumento no intervalo entre fabricação e uso proporciona um perda de informações sobre qualidade e possibilita maiores deteriorações ou perdas;
- *lead-times* extensos levam diretamente a proporcionalmente extensos inventários em processo;
- a tarefa de coordenação através dos itens para fabricação é usualmente feita mais dificilmente;
- estoques de segurança são afetados de duas formas. Primeiro eles devem proteger contra longos *lead-times*. Segundo, a variação da programação é tão grande quanto o período de programação. Então, para o estoque de segurança pode-se considerar que cresça proporcionalmente com o *lead-time*;
- a competitividade da empresa pode ser erodida pela pobre responsividade e datas de conclusão longas, derivadas de longos *lead-times*.

Conforme apresentado, os impactos resultantes da redução dos tempos de *setup* apresentam ramificações para todas as funções de custo e atendimento à demanda, com reflexo direto na sobrevivência da empresa. No entanto, esses benefícios devem ser cuidadosamente mensurados, pois podem ter reflexos indesejados em determinados ambientes.

Os investimentos em redução dos tempos de *setup* ocorrem quando a capacidade é uma restrição à saída do sistema, sendo que essa ação pode efetivamente aumentar a capacidade disponível pela redução do tempo total necessário à produção do mesmo lote de produção. Sendo assim, se esses investimentos forem feitos em caso em que a restrição de capacidade não for a principal restrição do sistema, esses investimentos gerarão capacidade produtiva ociosa e levarão a aumentos no custo total do sistema, considerando-se somente o aspecto de capacidade ociosa resultante do processo. No entanto, a redução dos

tempos de *setup* tem impacto também em volume de estoques e velocidade de atendimento à demanda, o que independentemente de capacidade ociosa leva a reduções de custos na empresa. Portanto, os cálculos de retorno sobre investimentos em redução de tempos de *setup* devem ser bem estruturados e levar em conta todos os aspectos envolvidos de uma forma geral na empresa, não se atendo somente aos custos em chão-de fábrica, mas englobando os custos de não atendimento à demanda e estoques oriundos da redução do *lead-time*.

2.3.3. Variação de Custos com a Redução do *Lead-Time* de Processo

Para o atendimento à demanda, a redução do *lead-time* de produção é ainda mais importante, pois possibilita um maior atendimento a demandas imprevistas. Com vistas ao cliente final, o atendimento a demandas imprevistas pode ficar comprometido principalmente quando o *lead-time* de produção for maior que o tempo necessário para o atendimento a essa demanda. Supondo que os estoques intermediários sejam reduzidos e não haja estoque final suficiente para atender à demanda, haverá a necessidade de produzir. Geralmente a flutuação da demanda é compensada com estoques, tanto de produtos acabados como de insumos, o que eleva os custos dos estoques e conseqüentemente os custos de operação. Nesse caso, com uma redução do *lead-time* de produção podemos ter uma condição mais favorável de atendimento à demanda sem a necessidade de recorrer à elevação dos níveis de estoque, e conseqüentemente evitar a elevação dos custos de produção.

Considerando somente uma função de custos do *setup*, dois podem ser associados com o estoque: custo fisico e financeiro. Para SPENCE & PORTEUS (1987), custos financeiros são aqueles que englobam os custos de oportunidade de capital transformado em estoque que é produzido antes de ser necessário, enquanto custos físicos são os custos que incorrem pela existência do estoque físico à mão (ou disponível).

Para o gerenciamento dos estoques, o impacto da redução no *lead-time* de produção ocorre na forma de que, para uma mesma demanda prevista, um menor *lead-time* de produção proporciona um maior giro dos estoques ao longo da empresa, pois permite uma redução nos níveis dos estoques iniciais e em processo. A

conseqüência é a redução nos custos relativos a esses estoques, como a administração dos estoques (estocagem e manuseio), custo de propriedade, obsolescência, seguro, danos e o custo de capital. A redução nos estoques e a conseqüente liberação de capital têm impacto direto na saúde financeira da empresa e em sua capacidade de reinvestimento, podendo essa redução de custos ser utilizada para a redução dos preços dos produtos ou simplesmente como aumento dos lucros. Pode-se dizer que *lead-times* de produção reduzidos são fundamentais para a redução dos custos com estoques e para o aumento do lucro financeiro.

CAPÍTULO 3

O PROBLEMA DE PROGRAMAÇÃO DE OPERAÇÕES *FLOW SHOP* PERMUTACIONAL COM TEMPOS DE *SETUP* ASSIMÉTRICOS E DEPENDENTES DA SEQÜÊNCIA

3.1. Contextualização do Sistema de Produção *Flow Shop* Permutacional

Em um sistema de produção, a definição de *flow shop* é dada por uma coleção de máquinas numeradas de tal forma que, para todas as tarefas a serem consideradas, a operação K é executada em uma máquina de maior numeração que a máquina da operação J, se $J<K$ (CONWAY, MAXWELL & MILLER, 1976). Em particular, cada operação depois da primeira tem exatamente um predecessor direto, e cada operação antes da última tem exatamente um sucessor direto. Então cada tarefa requer uma seqüência específica de operações a ser realizada para a tarefa ser completada. Este tipo de estrutura é chamado estrutura de precedência linear. O sistema de produção contém m diferentes máquinas e cada tarefa n consiste em m diferentes operações, cada uma das quais requerendo uma máquina diferente. O *flow shop permutacional* é caracterizado por um fluxo de trabalho unidirecional e contém uma ordem natural das máquinas, sendo que a quantidade de máquinas é igual ao número de operações das tarefas (BAKER, 1974).

A figura 4 apresenta, de uma forma geral, o fluxo de trabalho em um *flow shop* permutacional com m máquinas, evidenciando a entrada e saída de tarefas de forma linear e seqüencial.

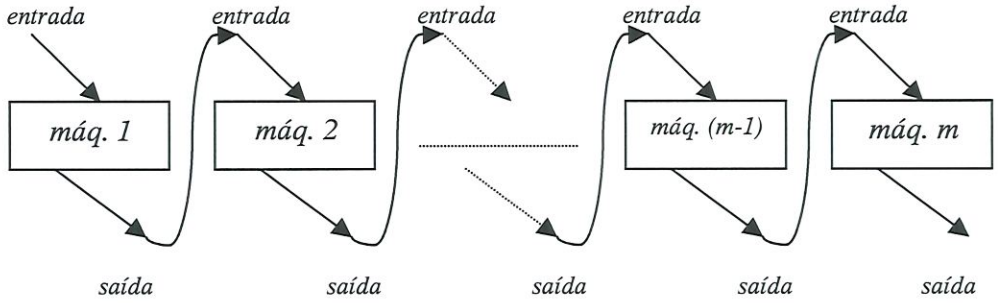


FIGURA 4: modelo geral de fluxo de trabalho em *flow shop* permutacional (adaptado de BAKER, 1974).

Considerando que as tarefas são numeradas por $J_1,J_2,....,J_n$, cada máquina por m_1, $m_2,....,m_x$ e que cada operação de cada tarefa é executada em uma determinada máquina e representada por (J_1,m_1), $(J_1,m_2),....(J_1,m_x)$, podemos considerar que as tarefas a serem executadas por uma mesma máquina $\boldsymbol{m_x}$ podem ser definidas por (J_1,m_x), $(J_2,m_x),....(J_n,m_x)$. Para cada operação $\boldsymbol{i}$ de uma determinada tarefa $\boldsymbol{J_j}$ (identificada por $\boldsymbol{J_{ji}}$) existe um tempo de processamento $\boldsymbol{P}$ (chamado $\boldsymbol{P_{Jji}}$). O critério de desempenho utilizado para esse trabalho é o de minimização do *makespan* ($\boldsymbol{C_{max.}}$), representado como segue:

$$C_{max.} = \sum_{x=1}^{m} \sum_{i=1}^{n} P_{Jjix} + \sum_{x=1}^{m} \sum_{i=1}^{n} I_{(z,y)ix}$$

onde: P_{Jj} = tempo de processamento da operação;

$I_{(z,y)}$ = tempo de espera entre operações;

Como representado, a minimização do *makespan* significa processar todas as tarefas programadas em todas as máquinas do sistema no menor tempo possível, considerando todos os tempos de processamento das $\boldsymbol{m}$ operações das $\boldsymbol{n}$ tarefas nas $\boldsymbol{m}$ máquinas do sistema mais os tempos de espera $\boldsymbol{I_{(z,y)}}$ entre tarefas (vide explicação detalhada após a figura 5). Para o modelo de *flow shop* permutacional tradicional, algumas considerações são feitas:

- os tempos de *setup* não são dependentes da seqüência e estão embutidos no tempo de processamento;
- m diferentes máquinas estão sempre disponíveis;
- uma tarefa J_y só pode começar a ser executada em uma máquina m_x após a operação ter sido concluída na máquina m_{x-1};
- as operações têm a mesma seqüência de máquinas para todas as tarefas, não sendo permitido adiantar operações, ou seja, há uma precedência tecnológica entre as operações;

A figura 5 apresenta, através do Gráfico de Gantt, o conceito do *makespan* e o carregamento de máquinas de um *flow shop* permutacional de 3 máquinas e 4 tarefas ***(3m x 4n)*** com as características apresentadas.

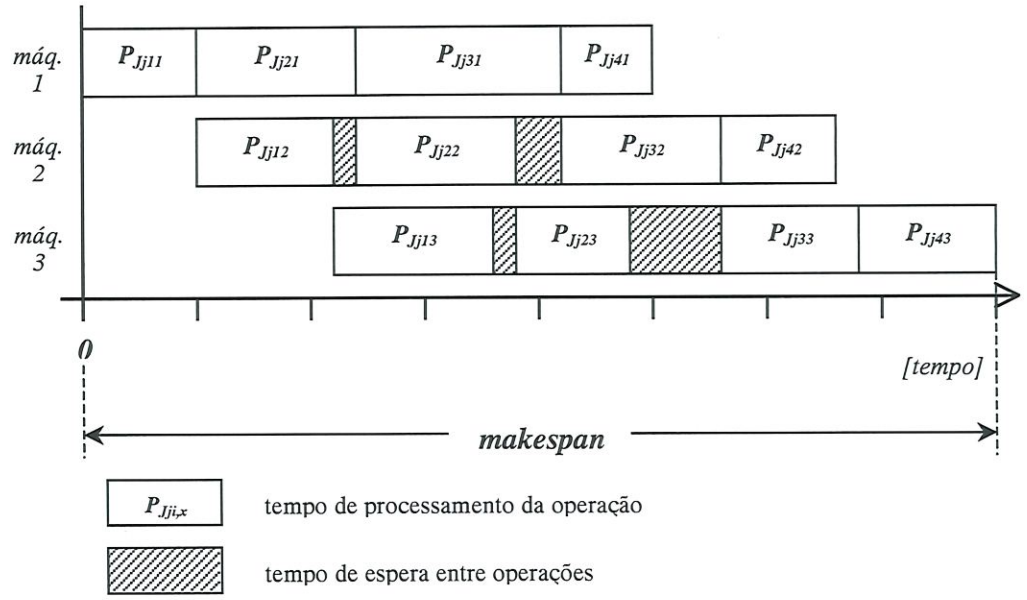


FIGURA 5: Gráfico de Gantt com exemplo de carregamento de máquinas em *flow shop* permutacional tradicional para um sistema de 3 máquinas e 4 tarefas (*3m x 4n*).

Os tempos de espera ($I_{(z,y)}$) podem ocorrer em função da condição de que uma operação de uma tarefa J não pode começar a ser executada em uma máquina m_x

se a operação anterior não tiver sido concluída na máquina m_{x-1}, e variam conforme o *set* de tempos de processamento e da seqüência estabelecida para as tarefas.

3.2. Caracterização do Problema de *Flow Shop* Permutacional com Tempos de *Setup* Assimétricos e Dependentes da Seqüência

3.2.1. O Conceito de *Flow Shop* Permutacional com Tempos de *Setup* Assimétricos e Dependentes da Seqüência

O *flow shop* permutacional com tempos de *setup* assimétricos e dependentas da seqüência *(ASDST)* se diferencia do *flow shop* tradicional pela abordagem em separado dos tempos de *setup*. Essa abordagem é necessária em função de sistemas de produção como, por exemplo, industrias química (tintas), onde o processo de limpeza é diferenciado considerando a cor de tinta que estava sendo produzido e a que será produzida. Em *ASDST* o tempo de *setup* entre operações de uma mesma tarefa *J*, identificado agora por $x_{a,b}$ (quando do *setup* da operação *a* para a operação *b*) em uma mesma máquina são independentes, são considerados em separado do tempo de processamento e por fim dependentes da seqüência estabelecida de tarefas conforme abaixo:

$$x_{a,b} \neq x_{a,c} \neq x_{b,c}$$

ou seja, o tempo de *setup* da operação *a* para a operação *c* é diferente do tempo de *setup* da operação *b* para a operação *c*.

Os tempos de *setup*, além de serem dependentes da seqüência também são assimétricos, conforme segue:

$$x_{a,b} \neq x_{b,a}$$

ou seja, o tempo de *setup* entre a operação *a* e a operação *b* é diferente do tempo de *setup* entre a operação *b* e a operação *a*.

A necessidade e importância embutidas nessa abordagem é que, por essas característica de assimetria e dependência da seqüência, há uma variação na soma

total dos tempos de *setup* em função do sequenciamento determinado às tarefas, levando à variação do critério de desempenho adotado, no caso o *makespan* ($C_{max.}$), conforme segue:

$$C_{max.} = \sum_{x=1}^{m} \sum_{i=1}^{n} P_{Jjix} + \sum_{x=1}^{m} \sum_{i=1}^{n} I_{(z,y)ix} + \sum_{x=1}^{m} \sum_{i=1}^{n} x_{(a,b)ix}$$

onde: P_{Jj} = tempo de processamento da operação;

$I_{(z,y)}$ = tempo de espera entre operações;

$x_{(a,b)}$ = tempo de *setup* entre operações.

O impacto do tempo de *setup* assimétrico e dependente da seqüência é, conforme já demonstrado, resultado da soma dos tempos de *setup* entre todas as ***m*** operações das ***n*** tarefas nas ***m*** máquinas. A figura 6 exemplifica uma matriz de tempos de *setup* em *ASDST* e a variação da somatória dos tempos em função da seqüência estabelecida de tarefas.

Matriz de tempos de setup para a máquina ***m***

	J1	*J2*	*J3*	*J4*
J1	---	7	33	27
J2	18	---	12	41
J3	25	17	---	13
J4	32	45	9	---

CONCEITO

- considerando-se a seqüência $S = \{J1, J2, J3, J4\}$, temos:

J1 *J2* *J3* *J4*

7 + 12 + 13 = 32

EXEMPLO 1

$S = \{J2, J1, J4, J3\}$

$\sum x_{(a,b)} = 18 + 27 + 9$

$\sum x_{(a,b)} = 54$

EXEMPLO 2

$S = \{J2, J4, J1, J3\}$

$\sum x_{(a,b)} = 41 + 32 + 33$

$\sum x_{(a,b)} = 106$

FIGURA 6: matriz de tempos de *setup* em *ASDST* e exemplos de variação de somatória em função da seqüência estabelecida.

Conforme apresentado, a somatória dos tempos de *setup* em *ASDST* varia conforme a seqüência estabelecida e, conseqüentemente, afeta diretamente o *makespan*. Esse comportamento é aletório e depende exclusivamente dos valores atribuídos à matriz de tempos de *setup*, e sua influência está ligada diretamente à relação de grandezas entre os tempos de *setup* e processamento, ou seja, quanto maiores os tempos de *setup* em relação aos tempos de processamento, maior a possibilidade de variação do *makespan*, já que os tempos de processamento são fixos.

A figura 7 apresenta, através do Gráfico de Gantt, o conceito do *makespan* e o carregamento de máquinas de um *flow shop* permutacional com *ASDST* de 3 máquinas e 4 tarefas ***(3m x 4n)***.

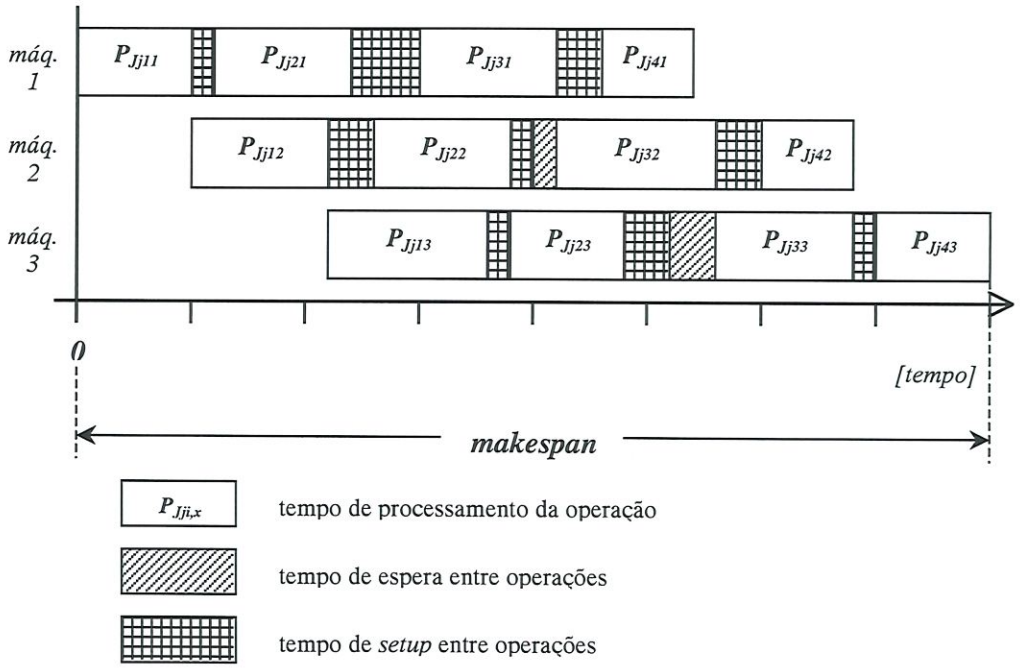


FIGURA 7: Gráfico de Gantt com exemplo de carregamento de máquinas em *flow shop* permutacional com *ASDST* para um sistema de 3 máquinas e 4 tarefas (*3m x 4n*).

Nesse exemplo, o *setup* de uma operação ***a*** pode ser executado na máquina ***m*** antes da mesma operação ***a*** estar concluída na máquina ***(m-1)*** e considera-se que o *setup* da primeira tarefa já esteja pronto, no tempo zero, para todas as máquinas. Essa

não é uma regra fechada para *ASDST*, devendo ser adaptada em condições reais conforme a necessidade e limitações do processo. Para o presente trabalho essa característica foi usada conforme o exemplo apresentado, por ser o mais usual encontrado na literatura pesquisada (RÍOS-MERCADO & BARD (1998), DAS, GUPTA & KUMAWALA (1995), SIMONS JR. (1992), RAJENDRAN & ZIEGLER (1997)) e também para que a base de comparação fosse mantida quando da comparação com o algoritmo proposto.

3.2.2. Pesquisas Desenvolvidas no Campo de *Flow Shop* Permutacional com Tempos de *Setup* Assimétricos e Dependentes da Seqüência

Para ALLAHVERDI, GUPTA & ALDOWAISAN (1999), a maioria das pesquisas em *scheduling* assume que os tempos de *setup* são desprezíveis ou parte do tempo de preparação. Essa afirmação simplifica a análise em certas aplicações, mais especificamente onde os tempos de *setup* são muito menores que os tempos de processamento, ou em casos de produções em lotes (*ou batch*) onde o *setup* é executado uma vez para um lote grande de produção. No entanto, para muitos casos há a necessidade de tratar os tempos de *setup* em separado sob risco de comprometer a qualidade da solução encontrada, principalmente nos problemas em que o tempo de *setup* é dependente da seqüência, sendo esse problema considerado como fortemente *NP-hard*.

O caso mais usual na literatura envolvendo tempos de *setup* dependentes da seqüência é o Problema do Caixeio Viajante (*TSP – Traveling Salesman Problem*), onde um caixerio viajante deseja sair de sua cidade e visitar cada *n-1* outras cidades, uma a uma, e retornar ao ponto de origem com objetivo de minimizar a distância total viajada (FLOOD, 1955 e LITTLE, MURTY, SWEENEY & KAREL, 1963), tendo sido proposto primeiramente por Hassler Whitney em 1934, em um seminário em Princeton. A aplicação do TSP em problemas de programação é quanto a máquinas simples, onde se tem um *set* de tarefas a serem seqüenciadas e diferentes tempos de *setup* entre suas operações.

KIM & BOBROWSKI (1994) realizaram experimentações computacionais comparando procedimentos para seqüenciamento com tempo de *setup* dependente da seqüência onde havia um tratamento particular para os tempos de *setup* e com tempos de *setup* incluídos no tempo de processamento. As comparações consideraram os níveis de estoque em processo (*WIP – Work In Process*), o custo total de sistema, o tempo de processamento total (*lead-time*), o inventário total, o atraso total e a eficiência do sistema. Concluíram que as regras envolvendo uma análise em separado dos tempos de *setup* incrementam a eficiência do chão-de-fábrica em mais de 4% relativo aos procedimentos com o tempo de *setup* incluso no tempo de processamento. De um modo geral, todos as medidas de desempenho observadas apresentaram melhor desempenho quando os tempos de *setup* são tratados em separado, sendo que essa melhoria observada cresce em função da razão entre tempos de *setup* e tempos de processamento. WILBRECHT & PRESCOTT (1969) experimentaram em uma grande companhia elétrica, cinco diferentes regras de programação de tarefas com diferentes medidas de desempenho. A primeira, chamada LOPN prioriza as tarefas com maior número de operações. A segunda, SIMSET, seqüencia as tarefas por similaridade entre *setups*. A terceira, OPNDD, prioriza as tarefas com data de encerramento mais próxima. A quarta, EXPED, sequencia pela prioridade de expedição e a quinta, HELP, prioriza as tarefas com maior numero de operações atrasadas. Dentre todas as regras, a que se destacou foi a SIMSET, que produziu o maior número de tarefas que deixaram a fábrica no período analisado. Esse resultado pode ser explicado em função de que a redução do tempo total despendido em *setups* aumenta a média de tempo disponível às máquinas transformarem peças e impacta diretamente no tempo médio de fluxo.

PATTERSON (1993) em um artigo chamado "*Analysis of Setup Time at Constraint Resource*" afirma que o campo de estudo em torno dos tempos de *setup* em um recurso gargalo é ainda inexplorado, e que a otimização dos tempos de *setup* via um procedimento para seqüenciamento é crucial para a otimização de programações. Apesar de não ser uma afirmação propriamente recente, em uma extensa busca na literatura poucas referências a procedimentos eficazes de otimização em *ASDST* são encontrados.

3.3. Métodos para a Solução do Problema de Programação de Operações *Flow Shop* Permutacional com Tempos de *Setup* Assimétricos Dependentes da Seqüência

3.3.1. Algoritmos para Máquina Única

ALLAHVERDI, GUPTA & ALDOWAISAN (1999) promoveram uma extensa revisão na literatura envolvendo as pesquisas em programação com tempos de *setup* em separado. Classificaram os problemas como em lotes (*batch*), não em lotes (*non-batch*), independente da seqüência, dependente da seqüência e classificaram a literatura de acordo com o ambiente de fábrica como máquina única, máquinas paralelas, *flow shops* e *job shops*, sumarizando os resultados das pesquisas correntes pelos diferentes tipos de problemas.

A abordagem inicial dos problemas com tempos de *setup* assimétricos dependentes da seqüência é com o Problema do Caixeiro Viajante (*TSP*), sendo que alguns métodos foram desenvolvidos com objetivo de resolver esse problema. LITTLE, MURTY, SWEENEY & KAREL (1963) e GARFINKEL (1973) desenvolveram algoritmos do tipo *branch-and-bound* buscando uma solução ótima para o problema de minimização do *makespan*. RUBIN & RAGATZ (1995) desenvolveram um algoritmo baseado em busca genética (*genetic search*) e compararam com um algoritmo tipo *branch-and-bound*, relatando resultados similares em termos de medida de desempenho adotada e superiores em eficiência computacional. KANELLAKIS & PAPADIMITRIOU (1980) desenvolveram um algoritmo heuristico baseado em busca local (*local search*), enquanto FISCHETTI & TOTH (1997) desenvolveram um algoritmo do tipo *branch-and-cut* para obter o melhor *tour* (circuito Hamiltoniano) e minimizar o custo total. GAVETT (1965), BAKER (1974) E CONWAY, MAXWELL & MILLER (1976) apresentam heurísticos baseados no *closest unvisited city* (ou "cidade mais próxima não visitada"), sendo que o algoritmo desenvolvido por GAVETT (1965) é uma adaptação do conceito e apresenta melhores resultados.

O trabalho mais referenciado na literatura é o de GAVETT (1965), em que apresenta o desenvolvimento de três heurísticos construtivos para seqüenciar tarefas

para um meio de produção simples a partir do *TSP* com objetivo de minimizar os tempos de espera no meio de produção ou o tempo de *setup* ao término de um lote finito de tarefas. O estudo examinou os três heurísticos em termos de dois critérios: primeiro frente ao tempo de espera ótimo obtido por um algoritmo *branch-and-bound* e segundo frente a uma seqüência randômica de tarefas através de um meio de produção. Os heurísticos foram denominados NB, NB' e NB'', sendo relatado que os dois primeiros têm desempenho ótimo com distribuição normal dos tempos de *setup* e o último com distribuição uniforme.

3.3.2. Algoritmos para *m* Máquinas e *n* Tarefas

Para os problemas com *m* máquinas e *n* tarefas alguns heurísticos construtivos foram desenvolvidos, com destaque aos algoritmos desenvolvidos por SIMONS JR (1992), usados como referência para outros algoritmos pela eficácia quanto a tempo de processamento e resultados obtidos. SIMONS JR. (1992) desenvolveu quatro heurísticos construtivos para o problema de *setup* assimétrico dependente da seqüência. O primeiro, chamado *MINIT* tem como objetivo a redução do tempo de espera, e o segundo, chamado *MICOT*, tem como objetivo a minimização do tempo total de processamento. Ambos partem da otimização da última máquina do sistema a ser seqüenciado e extendem o sequenciamento obtido para as máquinas restantes do sistema. Os outros dois, chamados *TOTAL* e *SETUP*, consistem na aplicação do Heurístico de Stinson para a matriz *n x n* (tarefa x tarefa) das tarefas do sistema. O *SETUP* usa a soma dos tempos de *setup* de todas as máquinas por célula da matriz, enquanto o *TOTAL* soma também os tempos de processamento das tarefas.

DAS, GUPTA & KHUMAWALA (1995) desenvolveram um algoritmo heurístico construtivo de *saving index* para minimizar o *makespan*. O algoritmo proposto determina os *savings* no tempo associado com uma seqüência particular e seleciona a seqüência com o máximo tempo salvo como a melhor solução heurística, tendo melhor performance com tempos de *setup* relativamente maiores que os tempos de processamento. Os *savings* são calculados através de fórmulas recursivas programando um a um os *sets* de tarefas do sistema partindo de uma máquina

randomicamente escolhida. O procedimento é repetido considerando como inicial cada máquina do sistema e os resultados comparados, sendo escolhida para resultado final a seqüência que representar o menor makespan.

RAJENDRAN & ZIEGLER (1997) desenvolveram um heurístico para minimizar a soma do tempo de fluxo ponderado de tarefas. O heurístico proposto usa como base o heurístico desenvolvido por DAS, GUPTA & KHUMAWALA (1995) alterando o critério de desempenho e o esquema de melhoria da seqüência inicial estabelecida.

PARTHASARATHY & RAJENDRAN (1997) desenvolveram um algoritmo heurístico baseado no *Simulated Annealing* com o objetivo de reduzir o atraso médio ao final de um lote finito de tarefas e conduziram um estudo de caso em uma industria fabricante de brocas. O algoritmo parte de uma solução inicial e a otimiza pela técnica de *Simulated Annealing*, usando um novo esquema de geração de vizinhança chamado Esquema de Perturbação por Inserção Randômica (*RIPS – Random Insertion Perturbation Scheme*). O critério de parada para o heurístico faz uso do valor de temperatura e o valor de congelamento contado, que é determinado pelo número de aceitação e seqüências geradas.

RÍOS-MERCADO & BARD (1998) apresentaram dois heurísticos construtivos com o objetivo de minimizar o *makespan*. O primeiro é uma adaptação do heurístico NEH incorporando o *setup* dependente da seqüência e chamado NEHT-RB. O outro é chamado GRASP (*Greedy Randomized Adaptive Search Procedure*) que é uma técnica que armazena bons resultados em uma variedade de problemas de otimização combinatorial. Adicionalmente desenvolveram procedimentos de busca local baseados no *TSP* (*Traveling Salesman Problem*) e adaptaram a cada um dos heurísticos.

CAPÍTULO 4

MÉTODO METAHEURÍSTICO *SIMULATED ANNEALING*

4.1. Aspectos Gerais

A técnica de *Simulated Annealing* tem como base uma analogia entre o resfriamento de um sólido e a otimização de um sistema com muitas variáveis independentes (PARTHASARATHY & RAJENDRAN, 1997). *Annealing* é o processo de aquecimento de um sólido a temperaturas muito altas até que as partículas, que estão randomicamente arranjadas na fase líquida são mudadas para atingir estados de baixa energia de estrutura regular, lentamente e com processo de resfriamento controlado. Em *Simulated Annealing* o sistema é lentamente resfriado até atingir a condição de congelamento. A correlação entre a metalurgia e os problemas de programação partiu de um trabalho chamado *Optimization by Simulated Annealing* publicado por KIRKPATRICK, GELATT & VECCHI (1983) onde houve a adaptação dos conceitos da metalurgia às técnicas de otimização.

O *Simulated Annealing* pode ser considerado como uma generalização do método descendente no qual a busca não estendida para um mínimo global é terminada após um mínimo local ser obtido, podendo ser classificado como um método heurístico melhorativo de busca aleatória na vizinhança (MOCCELLIN, 1994).

4.2. O Princípio de Funcionamento

O uso da técnica de *Simulated Annealing (SA)* é justificado pela capacidade de efetuar movimentos 'colina acima' na curva de soluções possíveis ao problema, explorando os 'vales' na tentativa de obtenção de uma solução ótima global. A partir de uma solução inicial obtêm-se seqüências derivadas através de um procedimento de busca na vizinhança. Se a seqüência derivada for superior à seqüência inicial pelos critérios de desempenho atribuídos, essa passa a ser a seqüência corrente. Caso contrário aplica-se a Distribuição de Boltzmann que calcula a probabilidade de encontrar-se uma melhor solução a partir de uma solução com um pior resultado (realizando o movimento chamado 'colina acima' ou *up hill*). Esse cálculo é realizado um número determinado de vezes (ou de iterações) o suficiente para percorrer a curva de soluções possíveis e encontrar a melhor solução global para o problema. Há uma variação na probabilidade de aceitação de uma solução pior, dinamicamente decrescente com o número de iterações. Para isso há um parâmetro de temperatura inicial e um de temperatura final, sendo que a probabilidade de aceitação é dada por um decréscimo concatenando o total de iterações, a temperatura inicial e a temperatura final.

4.3. Os Parâmetros Associados ao *Simulated Annealing*

A técnica de *Simulated Annealing* utiliza cinco parâmetros básicos para seu funcionamento, que são obtidos via literatura relatada ou adaptados aos algoritmos através de experimentação computacional. Abaixo a descrição e função de cada parâmetro:

- *TEMPERATURA INICIAL (T_1):* é o ponto de partida da aplicação do *SA*. OSMAN & POTTS (1989) desenvolveram uma fórmula para a temperatura inicial com variação através do número de tarefas e de máquinas conforme abaixo:

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

Como para o presente trabalho a aplicação da técnica é sobre uma matriz assimétrica somente de tarefas, a fórmula proposta não é aplicável. BUZZO & MOCCELLIN (1999) conduziram uma experimentação computacional analisando os resultados de um heurístico definido variando os valores de temperatura inicial. Constataram que os valores mais altos (60 e 70) têm melhor performance, pois permitem uma melhor flutuação entre o espaço de soluções por aceitar soluções piores;

- *TEMPERATURA FINAL(T_k):* a temperatura final é o estágio onde a aceitação de uma solução pior que a corrente é quase nula. MOCCELLIN (1994) realizou extensos experimentos computacionais considerando a temperatura final como 1, suportado por OSMAN & POTTS (1989);

- *NÚMERO TOTAL DE ITERAÇÕES (K):* define quantas vezes o método será repetido e entra no cálculo da função de resfriamento. O número de iterações varia conforme a matriz *n x m*, ou no caso apresentado, na matriz *n x n*. OSMAN & POTTS (1989) definiram uma fórmula para o cálculo como segue:

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

Como na temperatura inicial, a fórmula contempla tanto o número de máquinas *m* quanto o número de tarefas *n*. Para o presente trabalho, a determinação do número de iterações foi definido através de experimentação computacional, apresentada no item 5.3.

- *FUNÇÃO DE RESFRIAMENTO (β):* é o resultado concatenado entre as temperaturas inicial e final e o número de iteração. Define a taxa de decréscimo da temperatura corrente, e conseqüentemente a probabilidade de aceitação entre soluções comparadas. OSMAN & POTTS (1989) formularam a função de resfriamento conforme abaixo:

$$\beta = (T_1 - T_k) / ((K-1)T_1 * T_k)$$

- *PROBABILIDADE DE ACEITAÇÃO (p_k):* define se uma seqüência derivada S', pior que uma solução corrente S, é aceita como nova solução

corrente. O cálculo utiliza a Distribuição de Boltzman considerando-se a diferença entre o critério de desempenho adotado (Δ), conforme abaixo:

$$p_k = exp\ (-\Delta / T_k)$$

4.4. Estrutura Básica de Funcionamento do *Simulated Annealing*

O principio de funcionamento do *Simulated Annealing* é como segue:

- *PASSO 1:* determina-se uma solução inicial S, as temperaturas inicial e final e o total de iterações;
- *PASSO 2:* calcula-se β;
- *PASSO 3:* S" é a melhor seqüência entre todas as iterações e S''← S;
- *PASSO 4:* gera-se uma seqüência derivada S':
  - se S' < S, então S ← S';
  - se S' ≥ S, calcula-se p_k e um número randômico **R** entre 0 e 1;
    - se $p_k \geq R$, S ← S';
    - se $p_k < R$, S ← S;
  - se S ≤ S", então S''← S;
- *PASSO 5:* $T_{k+1} = T_k / (1 + \beta * T_k)$;
- *PASSO 6:* K = K-1;
  - se K > 0, volta ao PASSO 3;
  - se K = 0, S" é a melhor solução para o problema.

CAPÍTULO 5

O ALGORITMO PROPOSTO

5.1. Definição

O algoritmo proposto tem como objetivo minimizar o tempo total de processamento de *n* tarefas em *m* máquinas de um sistema de produção *flow shop (ou Makespan Total do Sistema)*, com tempos de *setup* assimétricos e dependentes da seqüência. A aplicação do algoritmo é baseada na otimização do recurso gargalo do sistema através da otimização de sua matriz de assimétrica de tempos de *setup*. Como isso se pretende diluir os tempos de espera entre tarefas nesse recurso para máquinas com tempos de processamento menor, buscando o equilíbrio do tempo de processamento das tarefas em cada máquina. O inicio da aplicação do algoritmo é através da identificação do principal recurso limitante à saída do sistema, caracterizado pelo recurso gargalo, como segue:

$$\sum_{i=1}^{n} p_{ig} = \underset{1<j<m}{\underline{max.}} \sum_{i=1}^{n} p_{ij}$$

onde p_{ig} é a representação dos tempos de processamento da tarefa i no recurso gargalo g e p_{ij} é representação dos tempos de processamento da tarefa i no recurso j.

Através do apresentado identifica-se, em função da matriz de tempos de processamento, o primeiro recurso gargalo que será o ponto de partida para a aplicação do algoritmo proposto. Sobre a matriz assimétrica de tempos de *setup* entre

tarefas desse primeiro recurso gargalo é feita a otimização para determinar a melhor seqüência para execução de todas as tarefas a serem programadas naquela máquina.

Para o algoritmo proposto, o *setup* entre um par de operações pode ser executado antes que a operação da tarefa seguinte a ser executada seja concluída na máquina anterior, ou seja, o *setup* é executado dentro de um tempo que possivelmente seria de espera. Após a execução do *setup*, se a tarefa ainda estiver em processamento na máquina anterior, então a máquina fica em espera aguardando sua conclusão.

Dada a nova abordagem a que o algoritmo proposto se dispõe, será chamado de "*Algoritmo Metaheurístico para Busca do Gargalo Flutuante com Simulated Annealing*", identificado a partir daqui por "***BGaFSA***".

A relevância da aplicação do *BGaFSA* é crescente com o aumento da razão entre os tempos de *setup* e o tempos de processamentos. Essa razão pode variar em função de um sistema de produção por lotes (*batch*) onde o *setup* é feito para uma família de produtos (a razão considerada é o tempo de *setup* dividido pela somatória dos tempos de processamento de todas as tarefas a serem executadas com o mesmo *setup*) e para tarefas individuais, onde a razão é calculada diretamente. O *BGaFSA* abrange ambas as possibilidades por ser baseado em um sistema assimétrico na matriz $x_{i,j}$ e que contempla tempos de *setup* iguais a zero. A aplicação do *BGaFSA* abrange desde uma única máquina ou meio de produção, até sistemas mais complexos com várias máquinas. O *BGaFSA* tem aplicação ótima para sistemas de produção com características de alta variedade / baixo volume de produção, pois a quantidade de *setups* executados tende a aumentar em função da própria característica do sistema e dos conceitos de redução de custos por redução de estoques, sendo o benefício direto da aplicação do algoritmo a redução do *lead-time* de produção das tarefas a serem executadas.

5.2. Obtenção da Solução Inicial

Para essa otimização parte-se de uma solução inicial obtida com a aplicação do algoritmo NB"(*Next Best Prime Rule*) desenvolvido por GAVETT (1965). O NB" é aplicado sobre a matriz assimétrica de tempos de *setup* entre tarefas subtraindo-se o

mínimo valor de cada coluna de todos os outros valores na coluna, considerando a similaridade entre os *setups*. O *tour* ótimo tem como resultado a menor soma de tempos de *setup* para a execução de todo o *set* de tarefas programadas. O algoritmo é baseado no *TSP* (*Traveling Salesman Problem*) e o resultado é circular e fechado, ou seja, o *tour* encontrado é ótimo para o processamento de todas as tarefas e a volta ao ponto de origem. Para o *BGaFSA* será considerada somente a execução do *set* de tarefas, sem volta ao ponto de origem. A escolha do NB" deve-se a ser um dos poucos heurísticos construtivos aplicáveis a um meio simples de produção para o caso assimétrico do *TSP*, e dentre os existentes ser considerado como o mais eficaz utilizando-se uma distribuição uniforme dos tempos de *setup* (GAVETT, 1965). O uso da solução inicial é baseado na otimização do tempo de processamento necessário ao *SA* encontrar uma melhor solução e também na eficácia e qualidade da solução final reportada por MOCCELLIN (1994), onde se observou uma melhora relativa nas soluções com o uso de uma solução inicial ao *SA*. A economia possível no tempo de processamento deve-se a que com a solução inicial, o procedimento de *SA* pode ser processado com uma quantidade menor de iterações, pois percorrerá o espaço de soluções possíveis de forma já otimizada.

5.3. Otimização com o *SA* e Definição do Número Total de Iterações

A otimização da seqüência inicial é obtida com a aplicação do método metaheurístico *Simulated Annealing*, partindo da solução inicial obtida com a aplicação do NB" e aplicando a ela uma técnica de obtenção de seqüências derivadas. A eficácia no uso da técnica de *Simulated Annealing* em problemas de tempos de *setup* dependentes da seqüência foi testada por TAN, NARASIMHAN, RUBIN & RAGATZ (2000), onde o *SA* foi comparado com algoritmos utilizando *branch-and-bound*, *random-start parwise interchange* e busca genética. Pelos resultados observados no estudo, as técnicas de *SA* e *random-start parwise interchange* apresentaram desempenho superior quanto à medida de desempenho adotada. A técnica de *Simulated Annealing* foi escolhida pela capacidade de aceitar soluções intermediárias piores que a corrente e então efetuar o movimento de *up hill*, conforme descrito no capitulo 4. Para o *BGaFSA*, o esquema de inserção utilizado foi

o *Shift Insertion* que obtém $(n-1)^2$ seqüências derivadas, conforme observa-se na figura 8.

	Seqüências Derivadas		
Seqüência Inicial	J_2, J_1, J_3, J_4	J_1, J_3, J_2, J_4	J_1, J_2, J_4, J_3
J_1, J_2, J_3, J_4	J_2, J_3, J_1, J_4	J_1, J_3, J_4, J_2	J_4, J_1, J_2, J_3
	J_2, J_3, J_4, J_1	J_3, J_1, J_2, J_4	J_1, J_4, J_2, J_3

FIGURA 8: seqüências derivadas de uma seqüência inicial obtidas pela aplicação do *Shift Insertion.*

Das seqüências derivadas obtidas, randomicamente escolhe-se uma que é comparada com a seqüência inicial, e é então aplicada a técnica do *Simulated Annealing (SA)*. O esquema de comparação é conforme o apresentado no *SA* e considera o critério de desempenho adotado.

Para determinar o número de iterações necessárias a cada *set* de análise em função do número de tarefas, foram executadas experiências computacionais e avaliados os resultados conforme abaixo:

TABELA 1: resultados da experimentação computacional para determinação do número médio de iterações para o SA.

CLASSE	ITERAÇÕES UTILIZADAS	MAIOR ITERAÇÃO	MENOR ITERAÇÃO	ITERAÇÃO MÉDIA
5 TAREFAS	10.000	4.200	0	1.435
10 TAREFAS	20.000	15.285	0	5.749
15 TAREFAS	28.000	24.564	0	7.596

Para a simulação foram utilizados altos valores de iterações para garantir a procura de uma solução ótima. Para a experimentação computacional do algoritmo são utilizados os valores máximos encontrados para efeito de garantir a otimização

com o *Simulated Annealing*, posto que o objetivo do trabalho não é avaliar o funcionamento do *SA*, e sim do algoritmo proposto. Quanto à coluna de menor iteração, devido à qualidade da solução inicial em alguns casos a aplicação do *Simulated Annealing* não melhora a solução inicial por já ser a melhor solução para o problema; daí estar como iteração 0 (zero). A análise dos resultados computacionais mostra que a quantidade de problemas em que a solução inicial não é melhorada cai com o aumento do número de tarefas, conforme observado quanto à qualidade da solução inicial (GAVETT, 1965).

5.4. Flutuação do Gargalo em Função da Otimização e Condição de Parada

Com a otimização da matriz assimétrica de tempos de *setup* da primeira máquina considerada como gargalo, calcula-se o *Makespan Total do Sistema* e verifica-se se não houve mudança do gargalo, ou seja, se o gargalo não passou para outra máquina. Essa verificação é feita através do cálculo do *makespan* por máquina, englobando os tempos de processamento, de *setup* e de espera entre tarefas (utilizado o Gráfico de Gantt para obter tais valores). Caso tenha ocorrido essa mudança, armazenam-se as informações de *Makespan Total do Sistema* e da seqüência utilizada para o seqüenciamento. Procede-se então ao recálculo em função da máquina identificada como novo gargalo até que o *loop* torne-se cíclico, ou seja, o gargalo retorne a uma máquina considerada anteriormente como gargalo. Essa é a condição de parada para o problema. Com a informação de *Makespan Total do Sistema,* a seqüência com o menor valor será a considerada ideal à resolução do sistema.

5.5. Seqüência Lógica do Algoritmo Proposto

1º. PASSO: na matriz de tempos de processamento das tarefas identificar a máquina-gargalo do sistema através de:

$$\sum_{j=1}^{n} p_{ig} = \underset{1 \le j \le m}{max.} \sum_{j=1}^{n} p_{ij}$$

2º. PASSO: na matriz assimétrica de tempos de *setup* da máquina-gargalo identificada no **1º. PASSO**, aplicar o heurístico construtivo NB"*(Next Best Double Prime Rule)* para determinar a seqüência inicial do problema;

3º. PASSO: aplicar o heurístico *Simulated Annealing* sobre a seqüência inicial. O resultado será a melhor seqüência obtida pela otimização da matriz de tempos de *setup* do recurso gargalo corrente;

4º. PASSO: considerar o seqüenciamento definido para o recurso gargalo, calcular o *Makespan* para todas máquinas de sistema separadamente e verificar se houve mudança do gargalo através da comparação do *Makespan* calculado por máquina;

5º. PASSO: se o maior valor de *Makespan* ocorrer em outra máquina do sistema que não seja o gargalo corrente, adotar essa máquina como novo gargalo, calcular o *Makespan Total do Sistema* e armazenar juntamente com a melhor seqüência obtida, e retornar ao **2º. PASSO,** processando a otimização sobre a matriz de tempos de *setup* do novo gargalo identificado. Se o maior valor de *Makespan* ocorrer no gargalo corrente ou em uma máquina anteriormente considerada como gargalo, encerrar o *loop* de cálculos e ir ao **6º. PASSO**;

6º. PASSO: dentre todos os *loops* das máquinas otimizadas, identificar a de menor *Makespan Total do Sistema.* Essa será considerada como novo gargalo do sistema e a seqüência obtida pela otimização de sua matriz de tempos de *setup* será a seqüência considerada ideal para todas as outras máquinas do sistema.

CAPÍTULO 6

ESTRUTURA DO *SOFTWARE* DESENVOLVIDO E COMPARATIVO ENTRE OS ALGORITMOS

6.1. Desenvolvimento do Programa

O algoritmo *BGaFSA* foi comparado com o algoritmo *TOTAL* desenvolvido por SIMONS JR. (1992) por ser um algoritmo bem referenciado na literatura (RÍOS-MERCADO & BARD, 1998 e ALLAHVERDI, GUPTA & ALDOWAISAN, 1999). Para a realização dessa comparação desenvolveu-se um software em linguagem DELPHI (versão 6) utilizado um micro-computador INTEL PENTIUM 233 MMX. Para o processo de comparação geraram-se matrizes randômicas em intervalos de dados determinados, resolvendo os mesmos *sets* de matrizes pelos dois algoritmos considerados. Os resultados obtidos com o *software* foram divididos em duas classes. A primeira classe de resultados é quanto a comparação do *BGaFSA* com *TOTAL,* e a segunda quanto aos aspectos da flutuação do gargalo no *BGaFSA*, ambas descritas em detalhes abaixo:

Primeira classe de resultados:

- melhor seqüência obtida com a aplicação de cada algoritmo e seus respectivos *makespan*;
- tempo de processamento de cada método;
- melhor algoritmo para as matrizes geradas e a melhora relativa porcentual entre os *makespan*.

Segunda classe de resultados:

- iteração do *SA* em que se obteve a melhor seqüência;
- posição dos gargalos inicial e final obtidos com a flutuação do gargalo no *BGaFSA*, seus respectivos *makespan* e a melhora relativa porcentual entre eles;

Na figura 9 é apresentada a tela principal do *software* desenvolvido:

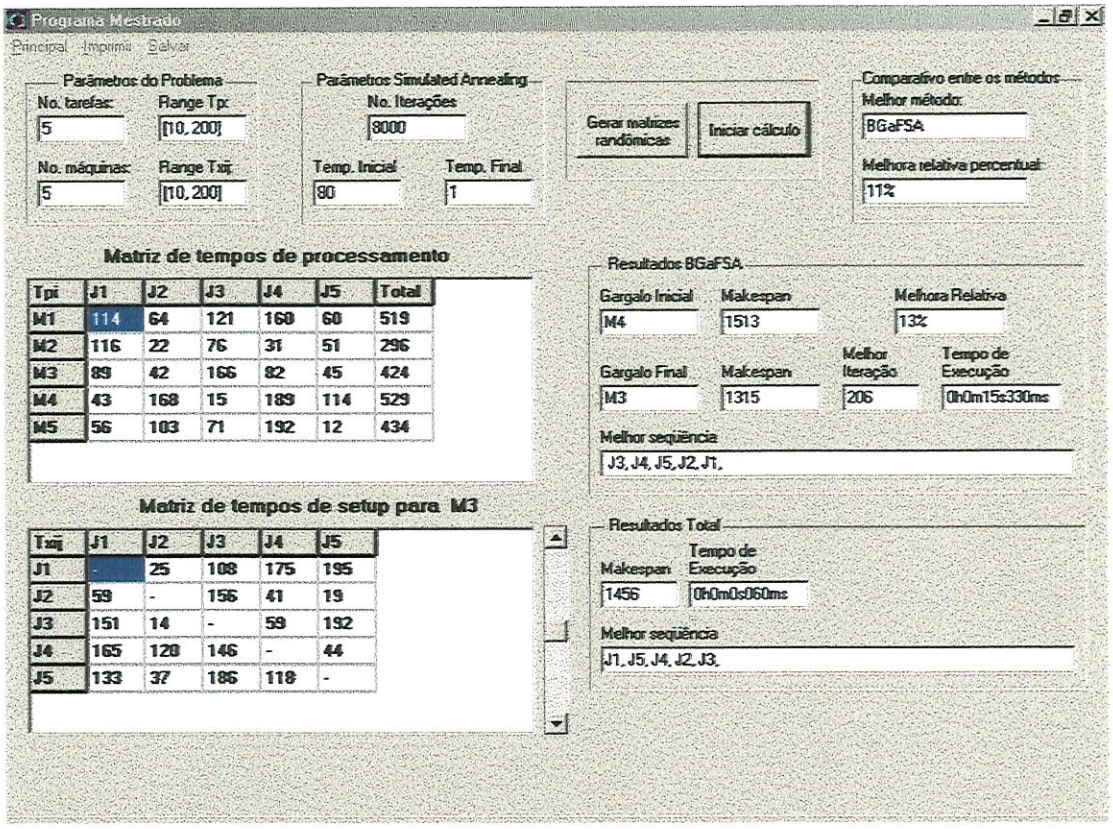


FIGURA 9: tela pricipal do *software* desenvolvido, com exemplo de problema resolvido.

Essas classes de resultados permitiram uma análise completa da abordagem do problema e seus reflexos, conforme apresentado no item 6.3 e capítulo 7. O *software* desenvolvido tem como opções:

- gerar matrizes randômicas à partir da definição do *range* dos tempos de *setup* e processamento (permite a determinação dos *ranges* de tempos);

- definir os parâmetros do *SA* (temperaturas inicial e final e número de iterações);
- imprimir resultados do comparativo (duas opções de impressão);
- montar Gráfico de Gantt com os resultados da aplicação dos dois algoritmos;
- apagar somente resultados (mantendo as matrizes geradas) ou problema inteiro (resultados e matrizes).

Na figura 10 é apresentado um exemplo da tela de gráficos, com Gráficos de Gantt, de um problema resolvido a partir da melhor seqüência obtida para cada um dos algoritmos considerados:

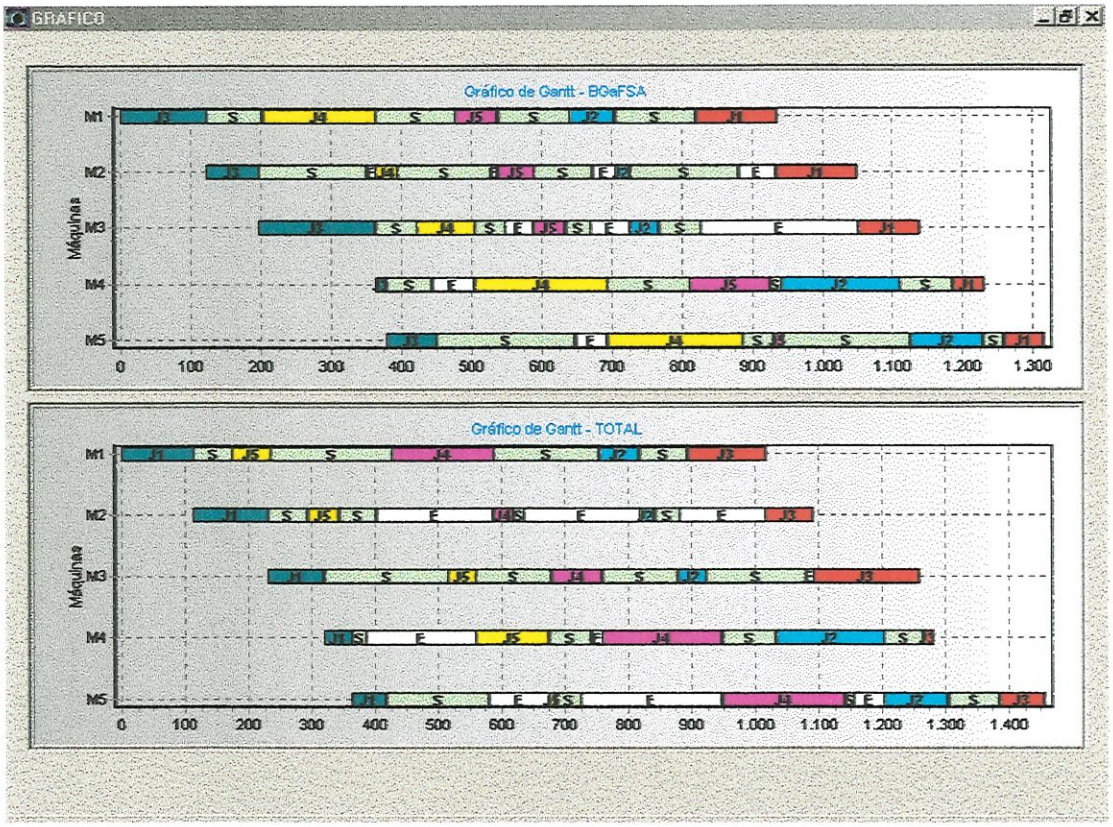


FIGURA 10: Gráficos de Gantt da tela de gráficos do *software* desenvolvido, com exemplo de um problema resolvido.

Para as opções *"Imprimir Resultados"*, os dados obtidos são impresso em forma de relatório e são conforme segue:

- imprimir as matrizes completas (tempos de processamento e *setup*);
- dados do problema (no. de máquinas e tarefas e *range* dos valores dos tempos de *setup* e processamento);
- parâmetros do *SA* utilizados (número de iterações e temperaturas inicial e final);
- primeira e segunda classes de resultados;
- imprimir Gráficos de Gantt.

Para a impressão são duas opções, com a diferença básica por uma ser um relatório completo contendo inclusive as matrizes dos problemas (tempos de processamento e *setup*) e a outra ser um modelo resumido, somente com os resultados e parâmetros dos problemas.

As opções do *software* permitem uma variação nos parâmetros dos problemas, possibilitando uma análise comparativa alterando-se os parâmetros inicialmente estabelecidos, por exemplo, para um mesmo *set* de matrizes geradas, comparando-se os resultados obtidos.

6.2. Medidas de Desempenho para o Comparativo entre os Algoritmos

Para a experimentação computacional foram gerados 180 problemas divididos em 9 classes de acordo com o número de tarefas (n) e máquinas (m) dentro de $nxm \in \{5, 10, 15\} \; x \; \{5, 10, 15\}$, onde para cada classe foram gerados 20 problemas. Considerando a literatura pesquisada quanto à determinação das classes, DAS, GUPTA & KHUMAWALA (1995) utilizaram $nxm \in \{5, 10, 15, 20\} \; x \; \{5, 10, 15, 20\}$, RAJENDRAN & ZIEGLER (1997) trabalharam com $nxm \in \{10, 15, 20, 25, 30\} \; x \; \{7, 30\}$ e RIOS-MERCADO & BARD (1997) com $nxm \in \{20, 50, 100\} \; x \; \{2, 4, 6\}$. Para a definição das classes para a aplicação do algoritmo proposto, a base utilizada foi o trabalho de SIMONS JR (1992) que utiliza $nxm \in \{5, 10, 15\} \; x \; \{5, 10, 15\}$ e que é a basa de comparação para o algoritmo proposto.

Para a comparação foram gerados problemas com tempos de *setup* e processamento identicamente distribuídos no intervalo [10, 200], na mesma base que o algoritmo de referência, e as matrizes resolvidas pelos dois algoritmos, sendo que todos os tempos foram gerados de forma randômica diretamente pelo *software* desenvolvido.

Os critérios para a comparação entre os algoritmos são os seguintes:

NÚMERO DE VITÓRIAS (OUTPERFORM): representa a quantidade de vezes que um algoritmo obteve um desempenho melhor que o outro, no critério de desempenho definido;

M.R.P.M.: representa a melhora relativa porcentual média entre os valores do critério de desempenho de cada algoritmo. A formula utilizada para o cálculo é como segue:

$$M.R.P.M.: \quad \frac{C_{max.1}}{C_{max.2}} * 100$$

T.M.P.: é o tempo médio de processamento de cada algoritmo para as mesmas matrizes geradas (ou mesmos problemas). A medição desse tempo é feita pelo *software*, e vai do início ao final das instruções para a execução de cada algoritmo.

Esses critérios foram definidos com base na literatura pesquisada, e utilizado de forma a lastrear a eficácia dos algoritmos. SIMONS JR. (1992) utiliza o critério de *M.R.P.M.* (denominado no trabalho por *Performance Index*), apresentando os resultados em termos de média e desvio padrão, com base no melhor resultado observado no *set* de tarefas analizado. DAS, GUPTA & KHUMAWALA (1995) apresentam resultados em estrutura similar a SIMONS JR (1992), aumentando a profundidade da abordagem ao incluir faixas específicas de análise por porcentual de desvio dos valores obtidos de *makespan* entre algoritmos comparados. RÍOS-MERCADO & BARD (1997) utilizam também o critéio de desvio médio, acrescentando a análise do tempo de processamento (*CPU time*) de cada algoritmo e PARTHASARATHY & RAJENDRAN (1997) acrescentam ao desvio médio a análise do tempo médio de atraso (*mean weighted tardiness*), que é o critério de

desempenho do algoritmo por eles desenvolvido. Dentre a literatura pesquisada em *ASDST* para *flow shop* permutacional, nenhum autor utiliza o critério de número de vitórias para o comparativo, ficando presos à análise do desvio médio. Isso pode mascarar a eficácia do algoritmo em termos de que um único bom resultado tratado de forma estatística pode compensar alguns resultados ruins. A opção por apresentar também a análise dos algoritmos comparados em termos de número de vitórias e comparar o comportamento com os resultados de desvio médio, procura fazer uma análise mais completa e entender os mecanismos de funcionamento do *BGaFSA*.

6.3. Comparativo entre *BGaFSA* e *TOTAL*

Os resultados obtidos com a aplicação do *software* foram tabulados a partir dos relatórios impressos, e o apresentado na tabela 2 é o resumo do desempenho computacional das medidas de desempenho estabelecidas, decorrente da comparação das soluções obtidas com a aplicação dos dois algoritmos.

TABELA 2: resumo do comparativo entre *BGaFSA* e *TOTAL*.

	BGaFSA			**TOTAL**		
CLASSE	**No. VITÓRIAS**	**M.R.P.M.**	**T.M.P. [s]**	**No. VITÓRIAS**	**M.R.P.M.**	**T.M.P. [s]**
5m x 5n	***17***	8,22%	15,83	3	3,03%	0,11
10m x 5n	***13***	7,23%	15,38	7	4,99%	0,17
15m x 5n	***15***	7,45%	17,10	5	7,09%	0,22
5m x 10n	***13***	5,29%	71,16	7	3,93%	0,30
10m x 10n	***11***	3,64%	98,76	9	4,88%	0,40
15m x 10n	***12***	6,19%	112,24	8	3,32%	0,50
5m x 15n	6	1,62%	206,76	***14***	8,31%	0,66
10m x 15n	5	2,80%	292,52	***15***	6,41%	0,75
15m x 15n	5	4,54%	405,58	***15***	2,84%	0,90

Os resultados realçados são os que os algoritmos tiveram desempenho superior em número de vitórias (*outperform*), considerado como o principal critério de desempenho. São apresentados a seguir os resultados obtidos com a experimentação computacional para cada critério, de forma gráfica e com considerações para cada item.

6.3.1. Comparativo quanto a Número de Vitórias

O número de vitórias é, conforme já apresentado, a quantidade de vezes em que um algoritmo obtém um resultado superior ao outro no critério de desempenho definido, no caso o *makespan*. A figura 11 demonstra a quantidade de vitórias do *BGaFSA* e do *TOTAL* para cada classe definida.

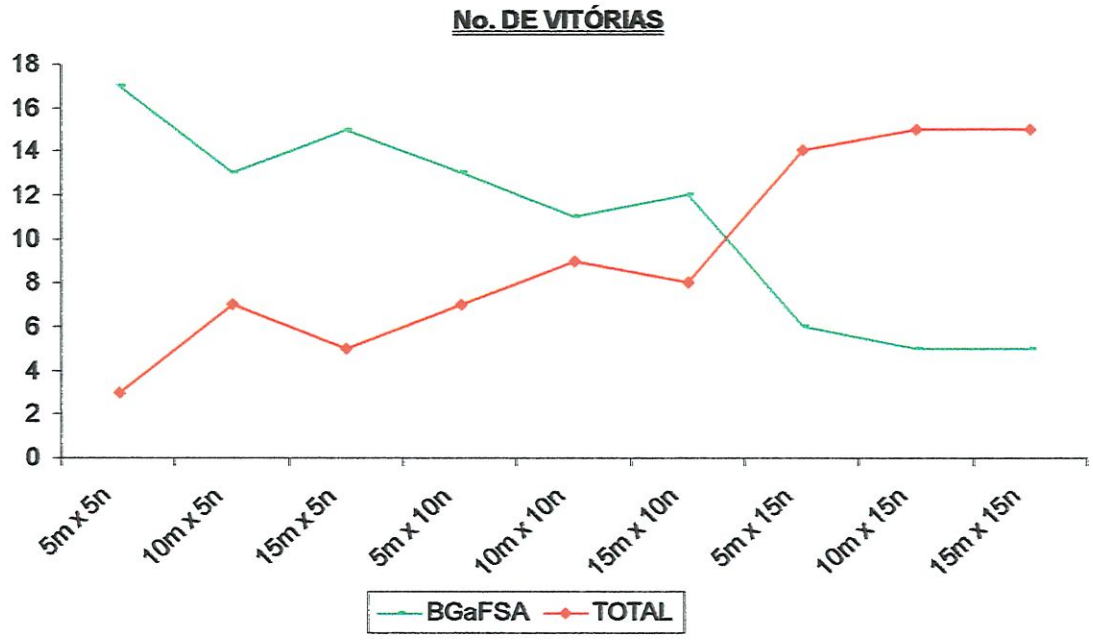


FIGURA 11: comparativo de número de vitórias entre *BGaFSA* e *TOTAL*.

Conforme apresentado, o *BGaFSA* obteve excentes resultados no comprativo com *TOTAL* em todos os problemas com 5 e 10 tarefas, mas com resultados inferiores nos problemas com 15 tarefas. As causas que levaram a esse comportamento têm relação provável com a otimização local obtida com a aplicação do *Simulated Annealing* e entra como proposta para pesquisas futuras (item 8.2).

6.3.2. Melhora Relativa Porcentual Média

A figura 12 demonstra a Melhora Relativa Porcentual Média (*M.R.P.M.*) e as linhas de tendências em função das classes observadas, obtidas com a experimentação computacional entre os dois algoritmos.

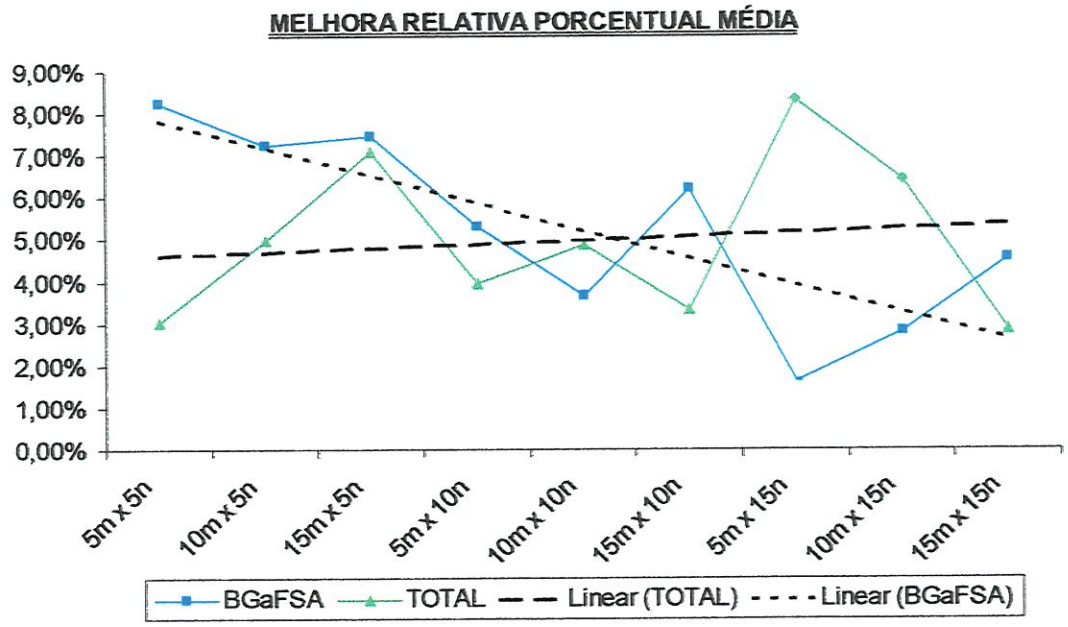


FIGURA 12: melhora relativa percentual média entre *makespan*, resultante da aplicação do *BGaFSA* e *TOTAL*.

Não se observa uma relação direta da *M.R.P.M.* e das linhas de tendência com as classes consideradas e nem com o número de vitórias de cada algoritmo, tendo sido observados inclusive classes em que um algoritmo obteve maior número de vitórias, porém o outro algoritmo obteve uma *M.R.P.M.* superior (classes de *10mx10n* e *15mx15n*). Conforme apresentado nas figuras 11 e 12 e em seus respectivos comentários, somente a análise conjunta dos resultados de número de vitórias e *M.R.P.M.* pode lastrar a eficácia de um algoritmo sem mascarar seu real comportamento. Como exemplo, para a classe de *15mx15n*, o *BGaFSA* obteve somente 5 vitórias em 20 problemas, enquanto o valor da *M.R.P.M.* foi superior aos resultados do algoritmo *TOTAL*.

Em uma análise específica das linhas de tendência observa-se que não há relação direta entre elas, pois não há complementariedade entre seus resultados, ou

seja, as inclinações das retas não são proporcionais. A conclusão para esse comportamento é que essa característica é aleatória e relativa somente ao *set* de matrizes analizado, independendo ainda do algoritmo considerado.

6.3.3. Análise do Tempo de Processamento dos Algoritmos

Conforme apresentado na tabela 2 e graficamente na figura 13, o *BGaFSA* apresentou tempos de processamento consideravelmente maiores que *TOTAL*. Essa diferença no tempo computacional deve-se a que a cada mudança no gargalo corrente o procedimento de *Simulated Annealing* é repetido buscando a otimização do gargalo corrente. Essa busca do gargalo pode ser repetida até *n* vezes, dependendo da composição das matrizes analisadas, em função da própria estrutura do algoritmo. A figura 13 mostra o comparativo entre o tempo médio de processamento para cada um dos algoritmos.

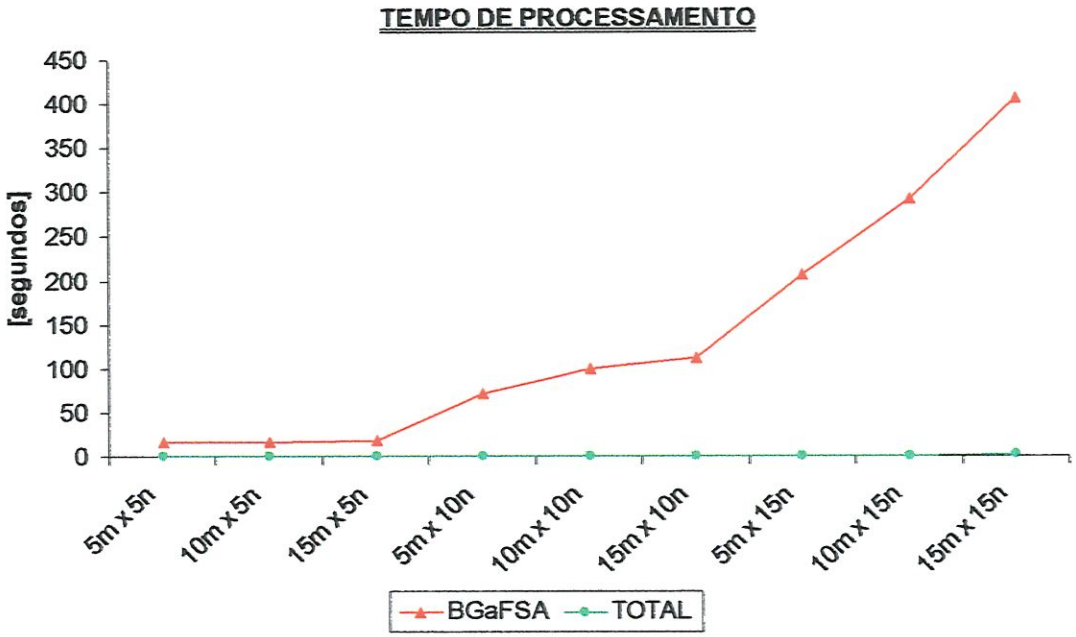


FIGURA 13: comparativo entre o tempo médio de processamento do *BGaFSA* e *TOTAL*, obtidos na experimentação computacional.

A eficiência computacional do algoritmo *TOTAL* é, conforme apresentado na figura 13, superior à eficiência do *BGaFSA*. Essa diferença também foi observada

por RÍOS-MERCADO & BARD (1995), e explicada como que para o algoritmo *TOTAL*, uma única e direta solução pode ser obtida para o mesmo set de matrizes, enquanto que para o *GRASP* de RÍOS-MERCADO & BARD (1995) e para o *BGaFSA* há parâmetros variáveis que podem levar a resultados diferentes para o mesmo set de matrizes e levam a uma maior complexidade dos algoritmos.

No entanto, considerando-se os tempos de processamento apresentados e a atual disponibilidade de recursos tecnológicos às empresas, o provável custo de processamento decorrente da aplicação do *BGaFSA* não é absurdo nem tão pouco impraticável em aplicações reais dada a complexidade do problema, sendo bem lastreado pela eficiência obtida em termos de número de vitórias nas classes consideradas.

CAPÍTULO 7

ANÁLISE DA FLUTUAÇÃO DO GARGALO NA APLICAÇÃO DO *BGaFSA*

7.1. Parâmetros Utilizados para a Análise da Flutuação do Gargalo

Devido ao ineditismo da abordagem feita pelo *BGaFSA* ao problema de *flowshop* permutacional com tempos de *setup* assimétricos e dependentes da seqüência, não há referências para lastrear a experimentação computacional e avaliar ou referenciar o comportamento do algoritmo proposto. Os dados e resultados apresentados neste capítulo partiram da observação e análise da aplicação do algoritmo com o *software* desenvolvido. As excessões são por conta das abordagens de número de iterações na aplicação do *Simulated Annealing* e variação no tempo de processamento do *BGaFSA*, ou sua eficiência computacional, tendo sido esses dois aspectos referenciados na literatura. Os parâmentros de observação e análise do algoritmo foram definidos como segue:

1. *OCORRÊNCIA DA FLUTUAÇÃO DO GARGALO E SEU SENTIDO DE FLUTUAÇÃO:* analisa a ocorrência da flutuação do gargalo e sua tendência de sentido, relativo ao gargalo inicial;
2. *FLUTUAÇÃO DO GARGALO VERSUS OTIMIZAÇÃO LOCAL:* analisa os ganhos obtidos com a flutuação do gargalo frente aos procedimentos usuais de seqüenciamento somente do recurso inicialmente identificado como gargalo;

3. *POSIÇÃO DO GARGALO FINAL NO SISTEMA:* analisa a tendência de posicionamento do gargalo na última máquina do sistema após a flutuação;

4. *NÚMERO DE ITERAÇÕES NA APLICAÇÃO DO SIMULATED ANNEALING:* analisa a quantidade de iterações necessárias à obtenção da solução ótima com a aplicação do *SA*;

5. *VARIAÇÃO NO TEMPO DE PROCESSAMENTO DO BGaFSA:* analisa as variações no tempo de processamento do algoritmo e suas causas

7.2. Ocorrência da Flutuação do Gargalo e seu Sentido de Flutuação

Um ponto observado é quanto à ocorrência da flutuação do gargalo a seu sentido de flutuação. Primeiramente, quanto à ocorrência da flutuação do gargalo, esse aspecto foi observado em 42,2% dos problemas gerados, conforme apresentado na figura 14, e constituem-se nos casos em que o gargalo final é diferente do gargalo inicialmente identificado. Já o gargalo estático são os casos em que o gargalo manteve-se o mesmo após a execução do algoritmo.

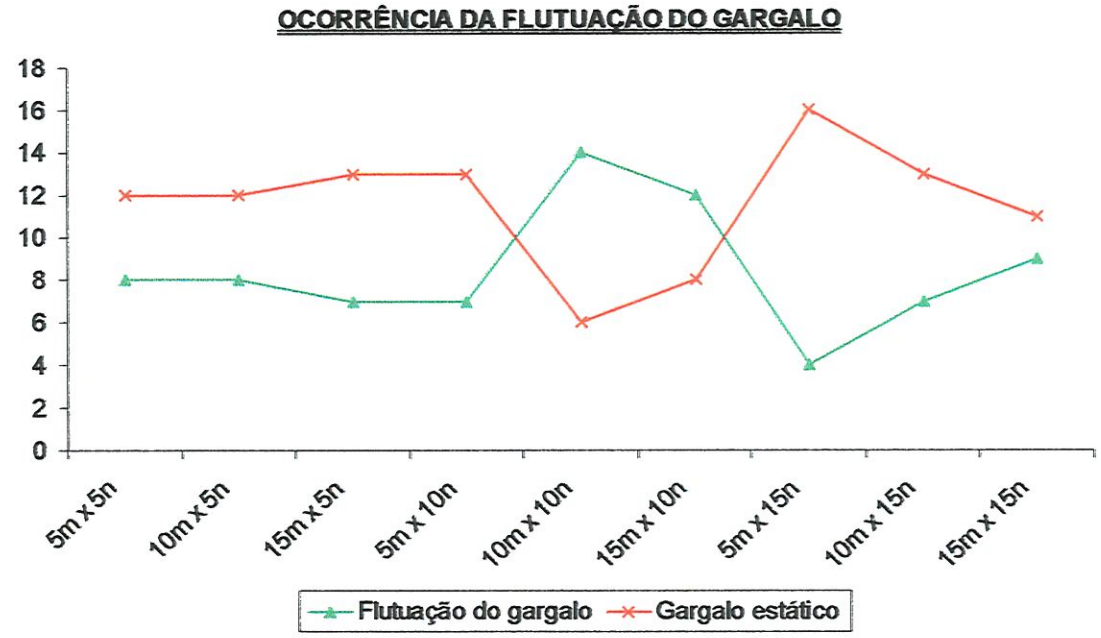


FIGURA 14: ocorrência da flutuação do gargalo em comparação com o gargalo estático.

A análise dessa flutuação não considerada as mudanças intermediárias do gargalo, onde o algoritmo considerou um gargalo corrente entre o gargalo inicial e final, mas o descartou em função do critério de desempenho definido. Pelo apresentado na figura 14, não há relação entre a ocorrência da flutuação do gargalo e o tamanho dos problemas, sendo sua ocorrência decorrente das matrizes consideradas.

Quanto ao sentido de flutuação do gargalo, na literatura pesquisada há algoritmos que consideram a otimização da última máquina do sistema como ideal para o sistema todo (SIMONS JR, 1992), ou seja, o gargalo tende a ser a última máquina do sistema pela suposta somatória dos tempos de espera entre tarefas na última máquina, que teoricamente tende a ser maior por arrastar os tempos de espera entre tarefas nas máquinas anteriores no sistema. No entanto, na experimentação computacional do *BGaFSA* observou-se que o gargalo pode não flutuar para baixo – para as últimas máquinas do sistema – e sim para uma máquina "acima" do gargalo inicial, ou seja, o gargalo "sobe". A tabela 3 mostra o sentido de flutuação do gargalo e a quantidade de problemas em que não houve mudança do gargalo inicial (gargalo estável), não tendo sido observado uma variação do sentido de flutuação do gargalo em função das classes consideradas. Observa-se, no entanto, uma constância quanto ao gargalo flutuar para cima, mas também sem relação com o tamanho do problema.

TABELA 3: porcentual relativo de sentido de flutuação do gargalo.

CLASSE	PARA CIMA	PARA BAIXO	ESTÁVEL
5m x 5n	0%	40%	60%
10m x 5n	5%	35%	60%
15m x 5n	5%	30%	65%
5m x 10n	5%	30%	65%
10m x 10n	5%	65%	30%
15m x 10n	5%	55%	40%
5m x 15n	0%	20%	80%
10m x 15n	0%	35%	65%
15m x 15n	15%	30%	55%

O encontrado quanto ao gargalo flutuar para cima é um dos pontos mais interessantes da aplicação do algoritmo, pois contraria o senso comum encontrado na literatura quanto à ocorrência do gargalo em sistema *ASDST*.

Pelo apresentado na figura 14 e na tabela 3 podemos afirmar que a ocorrência do gargalo e seu sentido de flutuação são aleatórios, independentes do tamanho do problema, variando somente com relação aos dados das matrizes geradas (tempos de processamento e *setup*).

7.3. Flutuação do Gargalo *Versus* Otimização Local

Outro ponto observado na aplicação do *BGaFSA* é quanto à melhora obtida no *makespan* entre a primeira máquina identificada como gargalo, ou gargalo inicial, e o gargalo identificado no final da aplicação, ou gargalo final. A figura 15 mostra a melhoria porcentual média obtida com o *BGaFSA* entre o *makespan* do gargalo inicial e do gargalo final.

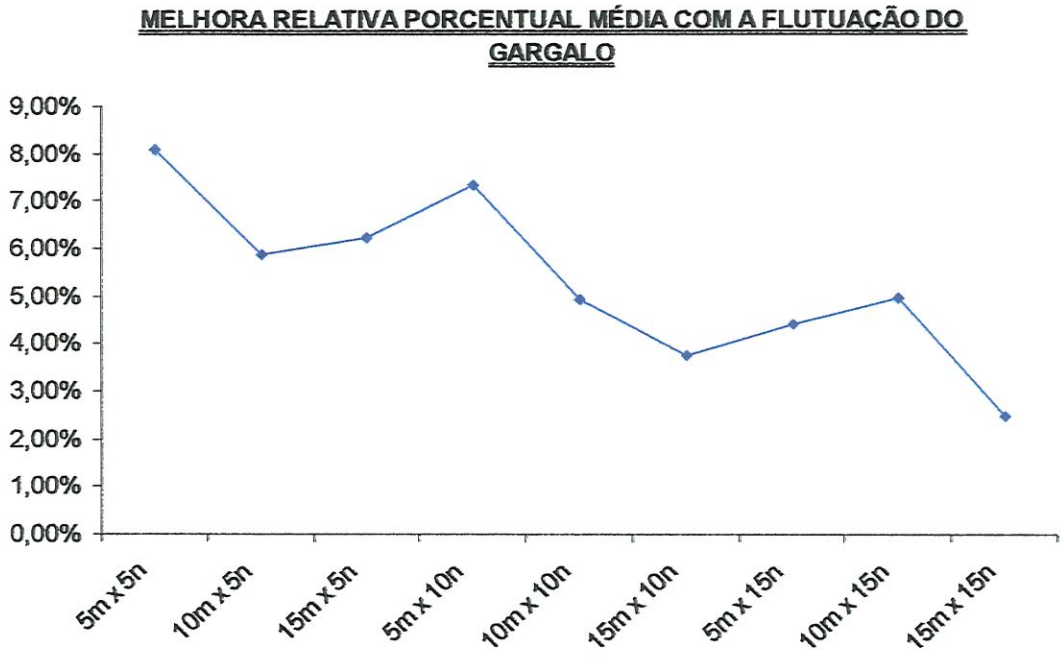


FIGURA 15: melhoria relativa porcentual média obtida com o *BGaFSA* entre o *makespan* do gargalo inicial e do gargalo final, considerando somente os problemas onde ocorreu a flutuação do gargalo.

Apesar da clara tendência de decrécimo do porcentual médio obtido com a melhora do *makespan* em função do aumento no tamanho dos problemas, não foi possivel identificar as causas que levaram a essa ocorrência. Na tabela 4 apresentam-se todas as faixas de variação do porcentual de melhoria obtido com a flutuação do gargalo.

TABELA 4: melhora obtida no *makespan* com a flutação do gargalo relativo à otimização do gargalo inicial.

CLASSE	MENOR VALOR	MAIOR VALOR	VALOR MÉDIO
5m x 5n	0,9%	17%	8,08%
5m x 10n	0,083%	12%	5,87%
5m x 15n	0,13%	16%	6,24%
10m x 5n	2,5%	13%	7,33%
10m x 10n	0,73%	14%	4,92%
10m x 15n	0,085%	7,7%	3,77%
15m x 5n	2,5%	5,9%	4,4%
15m x 10n	1,2%	9,3%	4,94%
15m x 15n	0,23%	4,3%	2,47%

A análise dos dados apresentados demonstra que a melhora obtida com a flutuação do gargalo teve impacto relevante nos problemas gerados quanto ao critério de desempenho adotado, sendo obsevado em alguns casos melhoras de até 17% no *makespan*.

7.3.1. Otimização Local e Flutuação do Gargalo *versus* TOTAL

Para reforçar o efeito da melhora relativa obtida com a flutuação do gargalo, a figura 16 mostra comparativamente o número de vitórias obtidas com a aplicação do *BGaFSA* em relação a *TOTAL* e o número de vitórias obtidas pelo procedimento de otimização local também frente a *TOTAL*. Observa-se que o conceito difundido na literatura quanto à otimização local no gargalo pode não ser o

suficiente para a obtenção de uma solução ótima ao problema, visto que a aplicação do *BGaFSA* obteve um número maior de vitórias em todas as classes analisadas, comparativamente com o procedimento de otimização local. Para se chegar a esse resultado, foram calculados os *makespan* do gargalo inicial e do gargalo final e comparados com o *makespan* obtido com *TOTAL*.

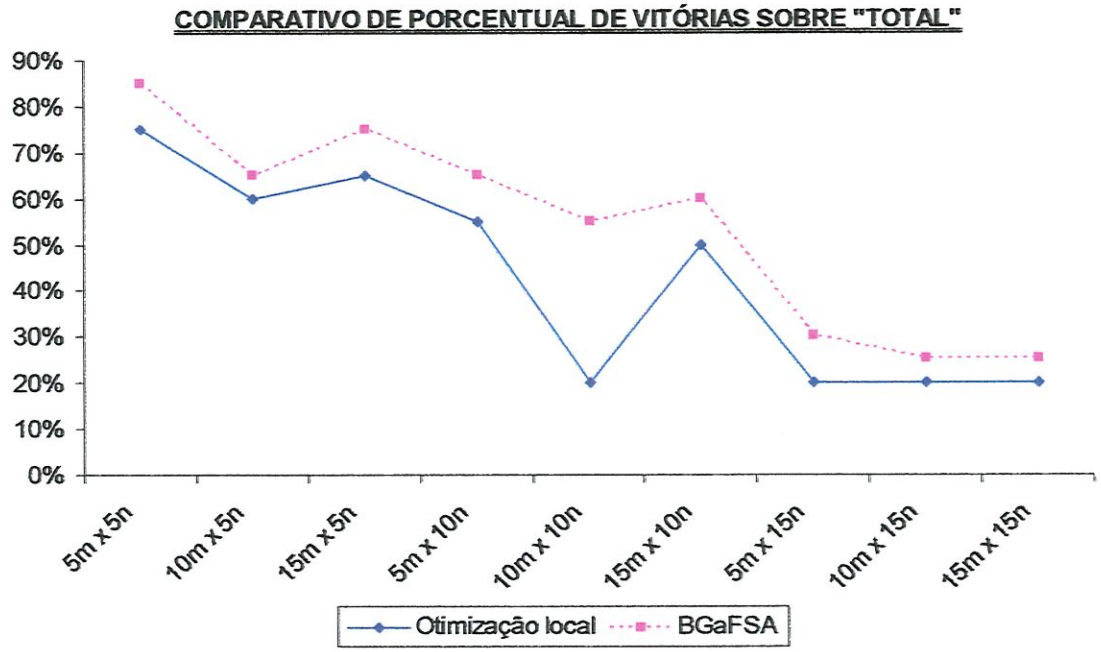


FIGURA 16: comparativo de percentual de vitórias sobre *TOTAL*, considerando a otimização local e o *BGaFSA*.

Conforme a figura 16 apresenta, em alguns casos se a otimização fosse somente local teria um resultado pior que *TOTAL*. A análise dos resultados obtidos com a aplicação do *BGaFSA* demonstra que algumas propostas usuais encontradas na literatura tendem a estar equivocadas. PINEDO (1995) propõe que se deve, seqüencialmente, identificar o recurso gargalo, sequenciar as tarefas nesse recurso e por último estender a seqüência obtida para os recursos não gargalo. Essa afirmação é similar ao procedimento de otimização local que, conforme demonstrado nas figuras 15 e 16 e também na tabela 4, tem eficiência inferior ao da aplicação do *BGaFSA*.

7.4. Posição do Gargalo Final no Sistema

Quanto à otimização da última máquina do sistema ser considerada como um procedimento ótimo para a solução do problema todo, a tabela 5 apresenta os resultados da análise da posição do gargalo final entre os problemas onde houve flutuação do gargalo.

TABELA 5: posição do gargalo final nos problemas onde houve flutuação do gargalo.

CLASSE	ÚLTIMA MÁQUINA	[1 a (n-1)] MÁQUINA
5m x 5n	75%	25%
10m x 5n	50%	50%
15m x 5n	14,3%	85,7%
5m x 10n	85,7%	14,3%
10m x 10n	64,3%	35,7%
15m x 10n	50%	50%
5m x 15n	50%	50%
10m x 15n	57,1%	42,9%
15m x 15n	22,2%	77,8%

Diferentemente do encontrado na literatura quando considera a otimização da última máquina como ideal para o restante do sistema, constatou-se que em média 48% dos problemas em que ocorreu a flutuação, o gargalo não se posicionou na última máquina do sistema. Observou-se também que, diferentemente do esperado, o percentual de ocorrências do gargalo na última máquina do sistema não se altera com o aumento do tamanho dos problemas em função do número de máquinas.

A análise apresentada fica mais clara quando observada a figura 17, onde fica evidente que não há tendência de variação da flutuação do gargalo em função das classes consideradas. Como já observado em outras características, essa varia somente conforme o *set* de matrizes analizadas, não variando nem conforme o número de máquinas (*m*) e nem conforme o número de tarefas (*n*).

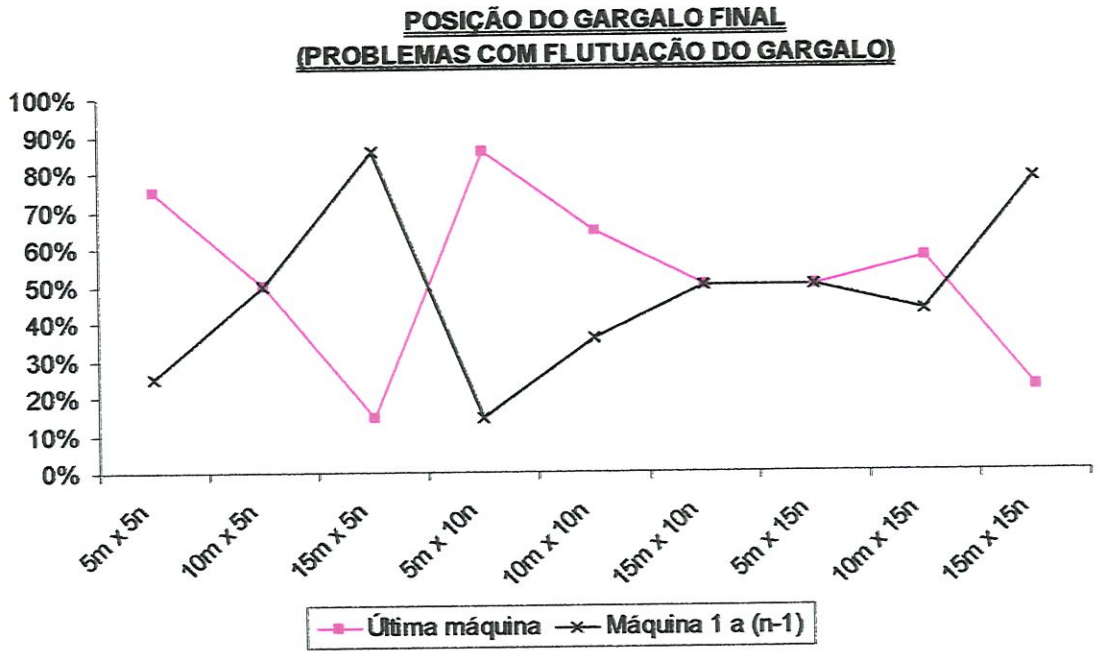


FIGURA 17: posição do gargalo final nos problemas em que ocorreu a flutuação do gargalo.

7.5. Variação no Tempo de Processamento do *BGaFSA*

A variação do tempo de processamento do *BGaFSA* varia diretamente com o número de vezes em que, durante seu processamento, há variação do gargalo corrente, sendo que a busca do gargalo pode se estender às (*n-1*) máquinas do sistema após a identificação do gargalo inicial, ou seja, o procedimento do *Simulated Annealing* pode ser repetido entre *1 e (n-1) vezes*, dependendo do *set* de matrizes analizado. Em função da imprevisibilidade dessa flutuação, o tempo de processamento da aplicação computacional do *BGaFSA* não é linear nem passível de se estabelecer uma função para determinação de sua eficiência computacional, pois para cada *set* de matrizes geradas um comportamento pode ser observado. A tabela 6 apresenta a variação nos tempos de processamento do *BGaFSA* obtidos com a experimentação computacional.

TABELA 6: variação dos tempos de processamento obtidos com a aplicação do *BGaFSA*.

CLASSE	MENOR TEMPO [s]	MAIOR TEMPO [s]	TEMPO MÉDIO [s]
5m x 5n	6,76	47,68	15,83
5m x 10n	6,37	29,16	15,38
5m x 15n	6,65	30,48	17,10
10m x 5n	31,75	163,79	71,16
10m x 10n	33,28	175,92	98,76
10m x 15n	53,45	180,00	112,24
15m x 5n	55,08	311,92	206,76
15m x 10n	89,14	523,44	292,52
15m x 15n	142,53	612,91	405,58

É observada uma dispersão bastante acentuada em todas as classes consideradas, sendo isso decorrente da própria estrutura do algoritmo. Uma proposta para futuras pesquisas, e que não foi objeto de estudo do presente trabalho, é esgotar os recursos do *Simulated Annealing* buscando que o método encontre a melhor solução na curva de soluções possíveis com um número menor de iterações. Isso representaria um incremento na eficiência computacional do algoritmo, pois reduziria o tempo necessário à execução de cada procedimento do *Simulated Annealing*.

7.6. Número de Iterações na Aplicação do *Simulated Annealing*

A análise do número de iterações na aplicação do *Simulated Annealing* demonstra o esforço computacional necessário à obtenção de uma solução ótima ao problema, estando ligado diretamente ao tempo de processamento ou esforço computacional necessário à execução do *SA*. Observa-se na figura 18 que o número médio de iterações varia conforme o número de tarefas, sendo estável com o número de máquinas. Conforme já apresentado, a análise do número de iterações ideal à

obtenção da melhor solução na curva de soluções possíveis não foi objeto de estudo deste trabalho, servindo como proposta para estudos futuros.

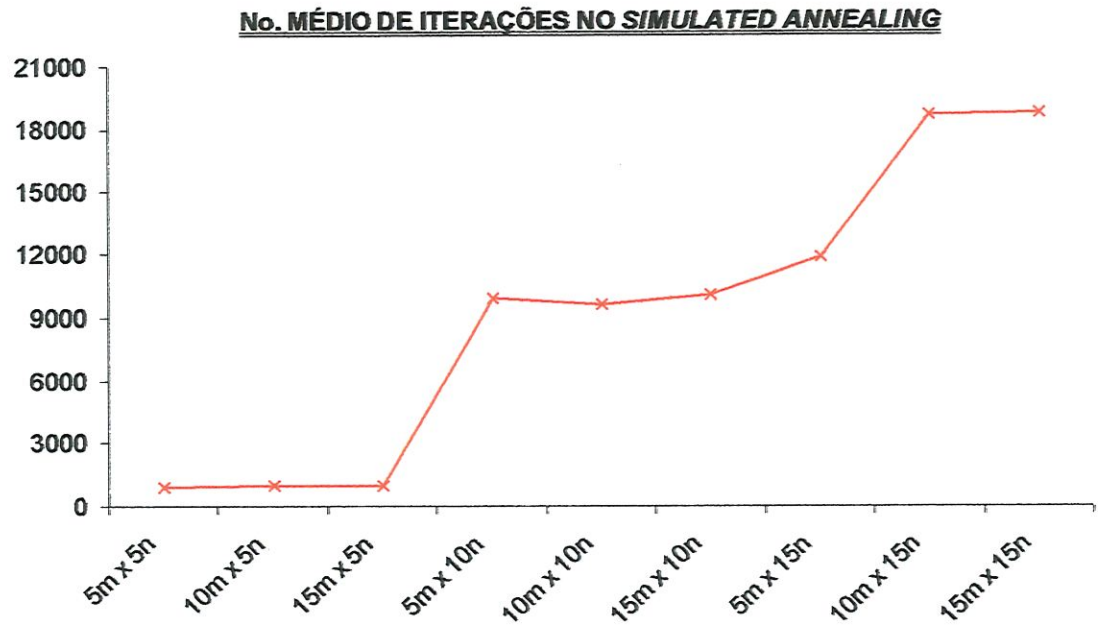


FIGURA 18: número médio de iterações do *Simulated Annealing* obtido com a aplicação do *BGaFSA*.

CAPÍTULO 8

CONCLUSÃO

8.1. Aspectos Gerais

Os resultados do comparativo entre os dois algoritmos considerados demonstram a eficácia do *BGaFSA* e lastreiam os resultados apresentados quanto à análise da ocorrência da flutuação do gargalo, seu sentido de flutuação e de sua posição de ocorrência após a flutuação. Demostram principalmente que a análise da flutuação do gargalo é um ponto muito importante a ser considerado na obtenção de uma solução ótima global para o sistema em *ASDST* e que essa flutuação ocorre e deve ser considerada. Essa afirmação fica evidenciada pelo apresentado em que o procedimento de otimização local apresentou resultados inferiores ao de busca do gargalo no comparativo com *TOTAL* (capitulo 7, item 7.3.1), e também quanto à melhora obtida entre o *makespan* do gargalo inicial e do gargalo final (capítulo 7, item 7.3). Ficou demonstrado ainda que a literatura em torno do problema de *ASDST* tende a se equivocar quanto a indicações de que a otimização local ou a otimização da última máquina do sistema como ideais para o sistema como um todo.

Na análise do comportamento do *BGaFSA*, observou-se que as características de ocorrência do gargalo, posição do gargalo e sentido de flutuação não têm uma ligação direta com o tamanho do problema (ou classe determinada), variando somente conforme o *set* de matrizes analisadas. Essa imprevisibilidade lastreia a eficácia e sugere o uso do *BGaFSA*, pois em um sistema real em *ASDST*, principalmente em condições de alto mix e baixo volume, a variação na composição das matrizes de tempos de processamento e *setup* são altas, podendo apresentar

comportamentos semelhantes ao observado nas experimentações computacionais apresentadas neste trabalho.

Em termos reais, a comprovação de que em um sistema *ASDST* o gargalo de produção pode não ser estático e varia conforme o *mix* de produção representa um complicador ao gerenciamento no chão-de-fábrica em termos de administração de recursos e programação do fator produtivo. Portanto, a administração do gargalo em um sistema *ASDST* tende a ser um processo extremamente complexo e necessita de um sistema estruturado de apoio e identificação dessa flutuação para possibilitar ações preventivas e garantir o atendimento à medida de desempenho adotada ao sistema. Dessa forma, uma fábrica com as características de produção em *ASDST* requer uma administração diferenciada das restrições e principalmente dos gargalos de produção, identificando dinamicamente esses gargalos, controlando e monitorando esses recursos de forma mais complexa e estruturada. A eficácia desse gerenciamento pode definir a vantagem competitiva da empresa frente aos concorrentes e ao cliente em termos de custos e atendimento à demanda.

8.2. Proposta para Pesquisas Futuras

Em função resultados apresentados da aplicação do *BGaFSA*, o algoritmo merece um estudo mais aprofundado de seu funcionamento. Alguns pontos do comportamento do *BGaFSA* necessitam de estudos mais aprofundados para permitir a compreenção da questão da flutuação do gargalo e promover uma melhor eficácia quanto a número de vitórias e tempo de processamento computacional. Aspectos de propostas para esses estudos são apresentados abaixo:

- *MELHORA NO DESEMPENHO DO MÉTODO HEURÍSTICO PARA CLASSES DE PROBLEMAS MAIORES:* uma alteração no método do *Simulated Annealing* ou a substituição por outro método de otimização pode permitir melhores resultados em termos de melhoria no desempenho do método proposto;

- *EFICIÊNCIA DE BUSCA NO ESPAÇO DE SOLUÇÕES:* um estudo mais aprofundado do comportamento do *Simulated Annealing* e seus parâmetros, principalmente em termos de número de iterações necessárias à busca de uma solução no espaço de soluções possíveis, pode promover uma melhora na eficiência de busca no espaço de soluções e conseqüentemente no tempo necessário ao processamento computacional do algoritmo.

Alterando-se esses parâmetros com sucesso, outros parâmetros associados ao comportamento do *BGaFSA* poderão ser alterados, arrastando a necessidade de revisão dos resultados aqui obtidos de forma a refletir os progressos obtidos.

ANEXO 1

PROGRAMAS COMPUTACIONAIS

IMPLEMENTAÇÃO COMPUTACIONAL

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, Grids, ExtCtrls, Menus, Math, Mxstore, MXDB,
  ComCtrls;

type
  TForm1 = class(TForm)
    EdX: TLabeledEdit;
    EdY: TLabeledEdit;
    Grid2: TStringGrid;
    Roller: TScrollBar;
    Label1: TLabel;
    Grid1: TStringGrid;
    MainMenu1: TMainMenu;
    Principal1: TMenuItem;
    Limparmatrizes1: TMenuItem;
    N1: TMenuItem;
    Sair1: TMenuItem;
    Label2: TLabel;
    Opes1: TMenuItem;
    EdTp: TLabeledEdit;
    EdTxij: TLabeledEdit;
    LbIteracoes: TLabeledEdit;
    BtnIniciarcalculo: TBitBtn;
    Grid3: TStringGrid;
    List1: TListBox;
    ParmetrosSA1: TMenuItem;
    Bevel1: TBevel;
    Bevel2: TBevel;
```

Label3: TLabel;
Label4: TLabel;
LblTempInicial: TLabeledEdit;
LblTempfinal: TLabeledEdit;
Grid4: TStringGrid;
GridInv: TStringGrid;
Grid5: TStringGrid;
GridSoma: TStringGrid;
Bevel3: TBevel;
BitBtngerarmatrizes: TBitBtn;
Bevel4: TBevel;
Bevel5: TBevel;
EdMak2: TLabeledEdit;
EdSeq2: TLabeledEdit;
Label5: TLabel;
Label6: TLabel;
EdTempo2: TLabeledEdit;
EdTempo: TLabeledEdit;
Edit2: TLabeledEdit;
Edit1: TLabeledEdit;
GridS: TStringGrid;
EdIteracao1: TLabeledEdit;
EdMaqGargalo: TLabeledEdit;
SaveDialog1: TSaveDialog;
OpenDialog1: TOpenDialog;
GridSoma2: TStringGrid;
EdGargaloInicial: TLabeledEdit;
Imprimirmatrizes1: TMenuItem;
Salvar1: TMenuItem;
Salvarproblemaresolvido1: TMenuItem;
Salvarmatrizes1: TMenuItem;
memo1: TRichEdit;
Bevel6: TBevel;
Label7: TLabel;
LabeledEdit1: TLabeledEdit;

```
    LabeledEdit2: TLabeledEdit;
    Limparresultados1: TMenuItem;
    Abrirarquivo1: TMenuItem;
    Montargrafico1: TMenuItem;
    Imprimirgrfico1: TMenuItem;
    Mak2: TLabeledEdit;
    Lbmelhora: TLabeledEdit;
    procedure RollerChange(Sender: TObject);
    procedure Somador1;
    procedure Sair1Click(Sender: TObject);
    procedure Limparmatrizes1Click(Sender: TObject);
    procedure BtnIniciarcalculoClick(Sender: TObject);
    procedure Definematrizes;
    procedure Somador2;
    procedure BitBtngerarmatrizesClick(Sender: TObject);
    procedure aceitacao;
    procedure Simmons;
    procedure Somador3;
    procedure ParmetrosSA1Click(Sender: TObject);
    procedure Imprimirmatrizes1Click(Sender: TObject);
    procedure Limparresultados1Click(Sender: TObject);
    procedure Montargrafico1Click(Sender: TObject);
    procedure Imprimirgrfico1Click(Sender: TObject);

  private
    procedure Escrevecabecalho;
    procedure calculosgrafico;
    procedure calculosgrafico1;
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
  matriz: array of array of array of string[6];
  X, Y, Z, PosZ, Iteracoes1, ItTotal, gargalo, linhaMaior: integer;
  Pos1, Pos2, bestmachine: integer;
  matrizxy: array of array of real;
  estatica: array [1..200] of double;
  matrizminima: array of Real;
  matrizcandidata: array of array of byte;
  SolucaoInicial: array of integer;
  SolucaoTemp: array of integer;
  SolucaoMelhor: array of integer;
  SolucaoFinal: array of integer;
  calculo: array of real;
  tempoinicial, tempotemporario, tempomelhor, TIni, TFinal, TempCorr,
  tempogargalo, bestmakespan: real;
  Formato, minimostr: string;
  matrizgrafico: array of array of real;
  matrizinsertion: array of string;
  matrizsolfinaldetecta: array of boolean;
  matrizsimmons: array [1..100] of double;
  matrizauxiliar: array [1..100] of double;
  matrizresultsimmons: array [1..100] of string;
  matrizsolfinalmakespan: array of real;
  matrizsolfinaltours: array of array of integer;
  matrizsolfinalsimmons: array of array [0..1] of integer;
  valormaxim: double;
  resultsimmons: string;
  TempoV, TempoJ, ProResul, Tempo1V, Tempo1J, ProResul1: TDateTime;

implementation

uses Unit2;

{SR *.dfm}
```

```
procedure TForm1.Definematrizes;
var
IndiceZ, IndiceY, Indice: integer;
begin
X:= strtoint (edX.text);
Y:= strtoint (edY.text);
  begin
  Grid1.RowCount:= Y+1;
  Grid1.Colcount:= X+2;
  SetLength(matrizxy, Y);
  For Indice:= low (matrizxy) to high (matrizxy) do
  setLength (matrizxy[Indice], X);
  end;

  begin
  grid2.Colcount:=X+1;
  grid2.Rowcount:=X+1;
  roller.max:=Y-1;
  PosZ:=roller.Position;
  SetLength (matriz,Y);
  For IndiceZ:= low(matriz) to high (matriz) do
    setLength(matriz[IndiceZ],X);
    For IndiceZ:= low(matriz) to high (matriz) do
      For IndiceY:= low(matriz[IndiceZ]) to high (matriz[IndiceZ]) do
      setLength(matriz[IndiceZ, IndiceY],X);
  end;
    Label1.Caption:= ('Matriz de tempos de setup para M1');
 begin
  grid3.Colcount:= X+1;
  grid3.Rowcount:= X+1;
  SetLength(matrizminima, X);
  Grid4.ColCount:= X+1;
  Grid4.RowCount:= X+1;
  GridInv.ColCount:= X+1;
  GridInv.RowCount:= X+1;
```

```
  Grid5.ColCount:= X+1;
  Grid5.RowCount:= X+1;
  GridSoma.RowCount:= Y+2;
  GridSoma.Colcount:= X+2;
  GridS.Colcount:= X+1;
  GridS.Rowcount:= X+1;
  GridSoma2.ColCount:= X+2;
  GridSoma2.RowCount:= X+2;
  SetLength (SolucaoInicial, X);
  SetLength (SolucaoFinal, X);
  SetLength (SolucaoTemp, X);
  SetLength (SolucaoMelhor, X);
  SetLength (calculo, X-1);
  SetLength (matrizgrafico, Y);
  For IndiceY:= 0 to Y-1 do
   SetLength (matrizgrafico[IndiceY], (9*X+1));
  SetLength (matrizinsertion, (X-1)*(X-1));
  SetLength(matrizsolfinaldetecta, Y);
  SetLength(matrizsolfinalsimmons, X);
  setLength(matrizsolfinalmakespan, Y);
  SetLength(matrizsolfinaltours, Y);
  For IndiceY:= 0 to Y-1 do
   SetLength(matrizsolfinaltours[indiceY], X);
  end;
end;

procedure TForm1.RollerChange(Sender: TObject);
var
limpagrid, PosX, PosY: integer;
A:string;
begin
A:= inttostr (roller.Position+1);
Label1.Caption:= ('Matriz de tempos de setup para '+ ' M' + A);
   For PosY:= low(matriz[posZ]) to high (matriz[posZ]) do
    For PosX:= low(matriz[posZ,PosY]) to High (matriz[PosZ,PosY]) do
```

```
 begin
  If Grid2.cells[posX+1, PosY+1]='' then Grid2.cells[posX+1, PosY+1]:='0';
   Matriz[posZ, PosY, PosX]:= (grid2.cells[PosX+1, PosY+1]);
 end;
 For limpagrid:= 0 to Grid2.rowcount do
  Grid2.Cols[limpagrid].clear;
  PosZ:= roller.Position;
  For PosY:= low(matriz[posZ]) to high (matriz[posZ]) do
   For PosX:= low(matriz[posZ,PosY]) to High (matriz[PosZ,PosY]) do
   Grid2.cells[posX+1, PosY+1]:= (matriz[PosZ, PosY, PosX]);
   For PosX:= 1 to X do
    For PosY:= 1 to X do
    If PosX<>PosY then
     If strtofloat(Grid2.Cells[PosX, PosY]) < 10 then
     Grid2.cells[PosX, PosY]:= '0' + Grid2.Cells [PosX, PosY];
       For PosX:=1 to X do
  For PosY:= 1 to X do
     If PosX= PosY then
     Grid2.Cells [PosX, PosY]:= '-';
end;

procedure TForm1.Somador1;
var
PosX, PosY: Integer;
Somador: Real;
begin
 For PosY := 1 TO Y DO
 Begin
  Somador := StrtoFloat(Grid1.cells[1,PosY]);
  For PosX := 2 to X DO
  Begin
  Somador := Somador + StrtoFloat(Grid1.cells[PosX,PosY]);
  Grid1.Cells[X+1,PosY] := FloattoStr(somador);
  End;
 end;
```

```
 For PosX:= 1 TO X DO
 Begin
  Somador := StrtoFloat(Grid1.cells[PosX,1]);
  For PosY:= 2 to Y DO
  Begin
  Somador := Somador + StrtoFloat(Grid1.cells[PosX,PosY]);
  GridSoma.Cells[PosX,Y+1] := FloattoStr(somador);
  End;
 end;
end;

procedure TForm1.Somador2;
var
PosX, PosY: Integer;
Somador2: Real;
begin
 For PosY := 1 TO X DO
 Begin
  Somador2 := StrtoFloat(Grid5.cells[1,PosY]);
  For PosX := 2 to (X-1) DO
  Begin
  Somador2 := Somador2 + StrtoFloat(Grid5.cells[PosX,PosY]);
  Grid5.Cells[X,PosY] := FloattoStr(somador2);
  End;
 end;
end;

procedure TForm1.Somador3;
var
PosX, PosY: Integer;
Somador3: Real;
begin
  For PosY := 1 TO X Do
  begin
  somador3:=0;
```

```
    For PosX := 1 to X DO
      begin
       Somador3:= Somador3 + StrtoFloat(GridSoma2.cells[PosX,PosY]);
       GridSoma2.Cells[X+1,PosY] := FloattoStr(somador3);
      End;
     end;
  For PosX:= 1 TO X DO
  Begin
    somador3:=0;
      For PosY:= 1 to X DO
      Begin
      Somador3 := Somador3 + StrtoFloat(GridSoma2.cells[PosX,PosY]);
      GridSoma2.Cells[PosX,X+1] := FloattoStr(somador3);
      End;
  end;
end;

procedure TForm1.Escrevecabecalho;
var
PosX, PosY: integer;
begin
  Grid1.Cells[0,0] := 'Tpi';
  For PosX := 1 TO X DO
  Grid1.Cells[PosX,0]:= 'J' + InttoStr(PosX);
  For PosY := 1 TO Y DO
  Grid1.Cells[0,PosY]:= 'M' + InttoStr(PosY);
  Grid2.Cells[0,0] := 'Txij';
  For PosX := 1 TO X DO
  Grid2.Cells[PosX,0]:= 'J' + InttoStr(PosX);
  For PosY := 1 TO X DO
  Grid2.Cells[0,PosY]:= 'J' + InttoStr(PosY);
  Grid1.cells[(X+1),0]:= 'Total';
end;
```

```
procedure TForm1.Sair1Click(Sender: TObject);
begin
 close;
end;

procedure TForm1.Limparmatrizes1Click(Sender: TObject);
var
Indice, PosY, PosX, PosZZ: integer;
begin
 for Indice:=1 to X+1 Do
  begin
  Grid1.cols[indice].clear;
  Grid3.cols[indice].clear;
  Grid4.Cols[indice].clear;
  Grid5.Cols[indice].clear;
  GridInv.Cols[Indice].clear;
  GridS.cols[indice].clear;
  GridSoma.cols[indice].clear;
  GridSoma2.cols[indice].clear;
  end;
  List1.Items.clear;
  begin
  For PosY:= 1 to X do
     For PosX:= 1 to X do
        Grid2.cells[posX, PosY]:= '';
  For PosZZ:= 0 to Y-1 do
   For PosY:= 0 to X-1 do
    For PosX:= 0 to X-1 do
      matriz[PosZZ, PosY, PosX]:= '';
  end;
  EdX.Text:= '';
  EdY.Text:= '';
  Edit1.Text:= '';
  Edit2.Text:= '';
  EdIteracao1.Text:= '';
```

```
EdMak2.Text:= '';
Mak2.Text:= '';
EdMaqGargalo.Text:= '';
EdSeq2.Text:= '';
EdTempo.Text:= '';
EdTempo2.Text:= '';
EdTp.Text:= '';
EdTxij.Text:= '';
LabeledEdit1.Text:= '';
LabeledEdit2.Text:= '';
LbIteracoes.Text:= '';
LblTempfinal.Text:= '';
LblTempInicial.text:= '';
Label1.caption:= 'Matrizes de tempos de setup';
EdGargaloInicial.text:= '';
SetLength (SolucaoInicial, 0);
SetLength (SolucaoFinal, 0);
SetLength (SolucaoTemp, 0);
SetLength (SolucaoMelhor, 0);
SetLength (calculo, 0);
SetLength (matrizgrafico, 0);
SetLength (matrizinsertion, 0);
SetLength(matrizsolfinaldetecta, 0);
SetLength(matrizsolfinalsimmons, 0);
setLength(matrizsolfinalmakespan, 0);
SetLength(matrizsolfinaltours, 0);
resultsimmons:= '';
X:=0;
Y:=0;
Z:=0;
PosZ:=0;
Iteracoes1:=0;
ItTotal:=0;
gargalo:=0;
linhaMaior:=0;
```

```
  tempoinicial:=0;
  tempotemporario:=0;
  tempomelhor:=0;
  TIni:=0;
  TFinal:=0;
  TempCorr:=0;
  tempogargalo:=0;
  bestmakespan:=0;
  end;

procedure TForm1.BtnIniciarcalculoClick(Sender: TObject);
var
PosY, PosX, PosI, PosT, TX, TY, TestaMin, PonEstatica, Listaminimo: integer;
valormaximo, VariZ, dec1, variavel, variavel1, variavel3, prresult: real;
LinhaGrid1, ContY, ContX, Cleaner, Minimo, contagem, linhamaior1,
Contclean, PositZ, Iteracoes, contaitem, contaparalelo, loopctrl, cont: integer;
variavellida, testestr, Fimstr, Tempstr, Melhorstr, ultimainsertion: string;

label
salto;

begin
ItTotal:= strtoint (Inputbox ('Iterações',
      'Digite o total de iterações do problema', ''));
TIni:= strtoint (Inputbox ('Temperaturas', 'Digite a temperatura inicial', ''));
TFinal:= strtoint (Inputbox ('Temperaturas', 'Digite a temperatura final', ''));
LbIteracoes.Text:= floattostr (ItTotal);
LblTempInicial.text:= floattostr (TIni);
LblTempFinal.Text:= floattostr (TFinal);
If MessageDlg ('Deseja continuar?', mtConfirmation, [mbYes, mbNo], 0)= mrNo then
    begin
    Form1.Close;
    end;
Tempo1J:= now;
For PosX:=0 to Y-1 do matrizsolfinalmakespan[PosX]:= $FFFF;
```

```
For PosY:= 1 to Y do
begin
 If Grid1.cells[X+1, PosY]='' then Grid1.cells[X+1, PosY]:='0';
  estatica[posY]:= strtofloat(grid1.cells[X+1, PosY]);
end;
 LinhaMaior:=0;
 valormaximo:=maxValue (estatica);
  For LinhaGrid1:= 1 to Y do
   If valormaximo = strtofloat (Grid1.cells[X+1, LinhaGrid1]) then
    LinhaMaior:= LinhaGrid1;
    Linhamaior1:= linhamaior;
    EdGargaloInicial.text:= 'M'+ inttostr(linhamaior);
    Roller.Position:= LinhaMaior-1;
    matrizsolfinaldetecta[linhamaior-1]:= true;
 For PosX:=1 to X do
  For PosY:= 1 to X do
   Grid3.Cells[posX,PosY]:= Grid2.Cells[posX,PosY];
 For PonEstatica:= 1 to 200 do Estatica[PonEstatica]:=$FFFF;
  For PosX:= 1 to X do
   For PosY:= 1 to X do
    Begin
    If PosX<>PosY then
     Begin
     Estatica[PosY]:= strtofloat(Grid3.Cells[PosX,PosY]);
     Matrizminima[PosX-1]:= minValue (estatica);
     end;
    end;
    For Listaminimo:=0  to X-1 do
     List1.items.add(floattostr(Matrizminima[ListaMinimo]));
    For PosX:= 1 to X do
     For PosY:= 1 to X do
      If PosX<>PosY then
      Grid3.cells[PosX, PosY]:= FormatFloat(Formato, ((strtofloat
      (Grid3.cells[PosX,PosY])) - (matrizminima[PosX-1])));
 For PosI:= 1 to X  do
```

```
    Begin
     For PonEstatica:= 1 to 200 do Estatica[PonEstatica]:=$FFFF;
       For PosX:= 1 to X do
        If PosX<>PosI then
        Estatica[PosX]:= strtofloat(Grid3.Cells[PosX,PosI]);
      end;
  For PosX:= 1 to X do
  For PosY:= 1 to X do
  grid4.Cells[PosX, PosY]:= '99';
  minimo:= 0;
  TX:= 1;
  TY:= 1;
     For PosX:= 1 to X do
      For PosY:= 1 to X do
      GridInv.Cells[PosX, PosY]:= Grid3.Cells [PosX, PosY];
   For PosT:= 1 to X do
    Begin
    Grid4.Cells[1, TY]:= Inttostr (PosT);
    ContY:= PosT;
    TX:= TX+1;
For PosX:= 1 to X do
   Grid3.Rows[posx].clear;
    For PosX:= 1 to X do
     For PosY:= 1 to X do
     Grid3.Cells[PosX, PosY]:= GridInv.Cells [PosX, PosY];
     For TestaMin:= 1 to X do
     Grid3.Cells[PosT, TestaMin]:= 'X' + Grid3.Cells[PosT, TestaMin];
    For PosY:= 1 to X-1 do
     Begin
     List1.items.clear;
       For PosX:= 1 to X do
        Begin
         If PosX <> ContY then
          List1.Items.add(Grid3.Cells[PosX, ContY]);
        end;
```

```
    For PosX:= 1 to X do
     Begin
      If Grid3.cells[PosX, ContY]= List1.Items[0] then
       Begin
       Minimo:=PosX;
       end;
     Grid4.Cells[TX, TY]:= InttoStr(minimo);
    end;
   For TestaMin:= 1 to X do
    Grid3.Cells[minimo, TestaMin]:= 'X' + Grid3.Cells[minimo, TestaMin];
    ContY:= minimo;
    TX:= TX+1;
  end;
TX:= 1;
TY:=TY+1;
 end;
   For PosY:= 1 to X-1 do
    For PosX:= 1 to X do
    Grid5.Cells[PosY, PosX]:= Grid2.Cells[strtoint(Grid4.Cells[PosY+1,PosX]),
      strtoint (Grid4.Cells[PosY,PosX])];
    For PonEstatica:= 1 to 200 do Estatica[PonEstatica]:=$FFFF;
     For PosX:=1 to X do
      Estatica[PosX]:= strtofloat(Grid5.Cells[X,PosX]);
       For PosX:= 1 to X do
        If Grid5.Cells[X,PosX]= floattostr (MinValue (estatica)) then
        Minimo:= PosX;
        For PosY:= 0 to X-1 do
         SolucaoInicial[PosY]:= strtoint (Grid4.Cells[PosY+1, Minimo]);
         minimostr:='';
         tempoinicial:= minValue (estatica);
          For PosY:= 0 to X-1 do
          minimostr:= minimostr + ' J'+ Grid4.Cells[PosY+1, minimo] +
           ',';
        Edit2.Text:= minimostr;
        Edit1.Text:= floattostr(tempoinicial);
```

```
      ItTotal:= strtoint(LbIteracoes.text);
      dec1:= (TIni-TFinal)/ItTotal;
      TempCorr:= TIni;
      tempomelhor:= tempoinicial;
      For PosX:=0 to X-1 do
        solucaomelhor[PosX]:= solucaoinicial[PosX];
For Iteracoes:= 1 to Ittotal do
  begin
      Iteracoes1:= Iteracoes;
For Loopctrl:=0 to(((X-1)*(X-1))-1) do
Matrizinsertion[Loopctrl]:='';
MinimoStr:='';
For Loopctrl:=0 to X-1 do
 If SolucaoInicial[LoopCtrl]>= 10 then
  MinimoStr:= Minimostr + floattostr(solucaoinicial[loopctrl])
  else
  MinimoStr:= Minimostr + '0' + floattostr(solucaoinicial[loopctrl]);
ContaItem:=0;
ContaParalelo:= 0;
PosY:=1;
While PosY < (X+X) do
begin
 PosX:=1;
  while PosX < (X+X) do
 begin
  UltimaInsertion:= minimostr;
  Variavellida:= copy (UltimaInsertion, PosY, 2);
  Delete (ultimainsertion, PosY, 2);
  Insert(variavellida, ultimainsertion , PosX);
  If not ((Contaitem mod (X+1) = 0) or (Contaitem mod (X+1)= X)) then
  Begin
  MatrizInsertion[ContaParalelo]:= UltimaInsertion;
  ContaParalelo:= ContaParalelo + 1;
  end;
 ContaItem := ContaItem + 1;
```

```
 PosX:= PosX + 2;
 end;
 PosY:= PosY + 2;
 Cont:= cont+1;
end;
      Tempstr:= matrizinsertion[Random(((X-1)*(X-1))-1)];
      PosY:= 1;
      PosX:= 0;
      While PosY < X*2 do
       begin
       SolucaoTemp[PosX]:= strtoint(Copy(TempStr, PosY, 2));
       PosX:=PosX+1;
       PosY:=PosY+2;
       end;
      For PosY:= 1 to X-1 do
       calculo[PosY-1]:= strtofloat(Grid2.Cells[SolucaoTemp[PosY],
        solucaoTemp[PosY-1]]);
         Tempotemporario:= 0;
         For PosX:= 0 to X-2 do
          TempoTemporario:= TempoTemporario + calculo[PosX];
   end;
   For PosX:=0 to X-1 do
   matrizsolfinaltours[linhaMaior-1,PosX]:= solucaomelhor[PosX];
matrizsolfinalmakespan[linhaMaior-1]:= matrizgrafico[(Y-1),((9*X)-6)];
If matrizsolfinaldetecta[gargalo-1]=true then
   begin
   For PosX:= 1 to 200 do estatica[PosX]:=$FFFF;
   For PosX:= 0 to Y-1 do
    estatica[PosX+1]:= matrizsolfinalmakespan[PosX];
 bestmakespan:= minvalue(estatica);
    For PosX:= 0 to Y-1 do
     If matrizsolfinalmakespan[PosX]= bestmakespan then
       begin
       bestmachine:= PosX+1;
       EdMaqGargalo.text:= 'M' + inttostr (bestmachine);
```

```
        Roller.position:= bestmachine-1;
         For PosY:= 0 to X-1 do
          solucaomelhor[PosY]:= matrizsolfinaltours[PosX, PosY];
         For PosY:= 1 to X do
          begin
           Melhorstr:= Melhorstr + ' J'+ inttostr(SolucaoMelhor[PosY-1]) + ',';
           Edit1.Text:= floattostr(bestmakespan);
           Edit2.text:= melhorstr;
          end;
      end;
    end
  else
    begin
    linhamaior:= gargalo;
    end;
  For PosX:= 0 to Y-1 do
      If PosX = (linhamaior1-1) then
      Mak2.text:= floattostr(matrizsolfinalmakespan[PosX]);
Tempo1V:= now;
ProResul1:= Tempo1V - Tempo1J;
EdTempo.Text:= FormatDateTime ('h"h"m"m"s"s"zzz"ms"', (ProResul1));
Lbmelhora.Text:= floattostrF(((strtofloat(Edit1.Text)/
          strtofloat(Mak2.Text)-1)*100*(-1)),ffgeneral, 00, 00) + '%';

variavel:= strtofloat(Edit1.Text);
variavel1:= strtofloat(EdMak2.Text);
If variavel <= variavel1 then
  begin
  Labelededit1.text:= 'BGaFSA';
  variavel3:= (((variavel1/variavel)-1)*100);
  Labelededit2.text:= floattostrF(variavel3, ffgeneral, 00, 00)+'%';
  end
else
  begin
  Labelededit1.text:= 'Simmons Jr. - Total';
```

```
  variavel3:= (((variavel/variavel1)-1)*100);
  Labelededit2.text:= floattostrF(variavel3, ffgeneral, 00, 00)+'%';
  end;
end;

procedure TForm1.BitBtngerarmatrizesClick(Sender: TObject);
var
PosX, PosY, PosA, XY, X, Y, nRandom, C, D, E, F: integer;
Begin
      X:= strtoint(inputbox('Definir parâmetros do problema',
         'Digite no. de tarefas', '0'));
      Y:= strtoint(inputbox('Definir parâmetros do problema',
         'Digite no. de máquinas', '0'));
      EdX.Text:= inttostr (X);
      EdY.Text:= inttostr (Y);
XY:= X*Y;
C:= strtoint (inputbox('Definir range Tp', 'Valor inicial','0'));
D:= strtoint (inputbox('Definir range Tp', 'Valor final','0'));
E:= strtoint (inputbox('Definir range Txij', 'Valor inicial','0'));
F:= strtoint (inputbox('Definir range Txij', 'Valor final','0'));

 for nRandom:=1 to XY do
  begin
  randomize;
    for PosY:=1 to Y do
      for PosX:=1 to X do
      grid1.cells[posX, posY]:= inttostr (randomRange (C,D));
      EdTp.text:= '['+Inttostr(C) + ', '+ Inttostr(D)+ ']';
      EdTxij.text:= '['+Inttostr(E) + ', '+ Inttostr(F)+ ']';
  end;
 randomize;
 for PosA:= 0 to Y-1 do
  begin
    for posY:=0 to X-1 do
    for posX:=0 to X-1 do
```

```
    matriz[PosA, PosY, PosX]:= floattostr((RandomRange(E,F)));
  end;
   For PosY:= low(matriz[posZ]) to high (matriz[posZ]) do
    For PosX:= low(matriz[posZ,PosY]) to High (matriz[PosZ,PosY]) do
      Grid2.cells[posX+1, PosY+1]:=  (matriz[PosZ, PosY, PosX]);
  For PosX:=1 to X do
   For PosY:= 1 to X do
    If PosX= PosY then
    Grid2.Cells [PosX, PosY]:= '-';
  Case F of
  0..99: Formato:= '00';
  100..999: Formato:= '000';
  1000..9999: Formato:= '0000';
  else
  Formato:='000000';
  end;
end;

procedure TForm1.aceitacao;
var
delta, fator1, fator2, dec1: real;
PosX: integer;
melhorstr: string;
begin
 ItTotal:= strtoint(LbIteracoes.text);
 dec1:= (TIni-TFinal)/ItTotal;
 delta:= tempotemporario - tempoinicial;
 If delta>=0 then
   Begin
   Fator1:= Exp(-delta/TempCorr);
   TempCorr:= TempCorr - dec1;
   Randomize;
    Fator2:= random(1000000000)/1000000000;
     If Fator1>= fator2 then
       begin
```

```
          tempoinicial:= tempoTemporario;
          For PosX:=0 to X-1 do
             solucaoinicial[PosX]:= solucaotemp[PosX];
          end
         else
          begin
          tempoinicial:= tempoinicial;
          solucaoinicial:= solucaoinicial;
          end
      end
     else
       begin
       tempoinicial:= tempotemporario;
       For PosX:=0 to X-1 do
       solucaoinicial[PosX]:= solucaotemp[PosX];
       end;
  If tempoinicial< tempomelhor then
   begin
   tempomelhor:= tempoinicial;
   EdIteracao1.text:= inttostr(Iteracoes1);
     For PosX:= 0 to X-1 do
       solucaomelhor[PosX]:= solucaoinicial[PosX];
     For PosX:= 1 to X do
       begin
       Melhorstr:= Melhorstr + ' J'+ inttostr(SolucaoMelhor[PosX-1]) + ',';
       Edit1.Text:= floattostr(tempomelhor);
       Edit2.text:= melhorstr;
       end;
  end;
end;

procedure TForm1.Simmons;
var
PosY, PosX, PosA, PosD, PosB, PositZ, Pont, Pontei, Pont2, PosZZ, PonteiroX, VariZ:
integer;
```

```
GridBug: double;
K, L: string;

label
salto2, salto3, salto4;

begin
TempoJ:= now;

 For PosY:=0 to X-1 do
  For PosX:=0 to X-1 do
   Begin
    VariZ:=0;
    For PositZ:= 0 to Y-1 do
     If PosX=PosY then
       GridS.Cells[PosY+1, PosX+1]:='-'
     else
       Begin
       VariZ:= VariZ + strtoint(matriz[PositZ, PosY, PosX]);
       GridS.Cells[PosX+1, PosY+1]:= floattostr (VariZ);
       end;
   end;
GridSoma2.cells[X+1, X+1]:='X';
For PosY:= 0 to X-1 do
 For PosX:= 0 to X-1 do
  If PosX<>PosY then
  GridSoma2.cells[PosY+1,PosX+1]:= inttostr(strtoint(GridS.cells[PosY+1,PosX+1])+
      strtoint(GridSoma.cells[PosY+1,Y+1]))
  else
  GridSoma2.cells[PosY+1, PosX+1]:='X';
For PosX:= 0 to 1 do
 For PosZZ:= 1 to X do
  matrizsolfinalsimmons[PosX, PosZZ]:=0;
For PonteiroX:= 1 to X do
begin
```

```
 List1.Sorted:= true;
For PosX:= 1 to X do
  begin
    List1.Items.clear;
    For Pont:=1 to X do
    If GridSoma2.Cells[PosX,Pont]<>'X' then
    List1.Items.add(GridSoma2.cells[PosX,Pont]);
     If List1.count=0 then goto Salto2;
     If List1.count>1 then
       begin
       K:= (List1.items[1]);
       L:= (List1.items[0]);
       GridSoma2.Cells[PosX, X+1]:= formatfloat(Formato, ((strtofloat(K) - strtofloat(L))));
       end
    else
       begin
       L:= (List1.items[0]);
       GridSoma2.Cells[PosX, X+1]:= formatfloat(Formato, (strtofloat (L)));
       end;
   salto2:
 end;
For PosX:= 1 to X do
 begin
 List1.Items.clear;
  For Pont:=1 to X do
   If GridSoma2.Cells[Pont, PosX]<>'X' then
   List1.Items.add(GridSoma2.cells[Pont, PosX]);
    If List1.count=0 then goto Salto3;
     If List1.count>1 then
       begin
       K:= (List1.items[1]);
       L:= (List1.items[0]);
       GridSoma2.Cells[X+1, PosX]:= formatfloat(Formato, ((strtofloat(K) - strtofloat(L))));
       end
     else
```

```
        begin
        L:= (List1.items[0]);
        GridSoma2.Cells[X+1, PosX]:= formatfloat(Formato, (strtofloat (L)));
        end;
end;
For PosX:= 1 to 100 do
matrizsimmons[PosX]:= 0;
For Pont2:= 1 to X do
 begin
 If GridSoma2.cells[X+1, Pont2]<>'X' then
 matrizsimmons[Pont2]:= strtoint(GridSoma2.cells[X+1, Pont2]);
 end;
For Pont2:= 1 to X do
 begin
 If GridSoma2.cells[Pont2, X+1]<>'X' then
 matrizsimmons[(Pont2)+(X)]:= strtoint(GridSoma2.cells[Pont2, X+1]);
 end;
valormaxim := maxValue(matrizsimmons);
Pos1:=0;
Pos2:=0;
For PosX:= 1 to X+1 do
 For PosY:= 1 to X+1 do
  If GridSoma2.cells[PosX, PosY]<>'X' then
  begin
  gridBug:= strtofloat(GridSoma2.cells[PosX, PosY]);
  If gridbug= valormaxim then
   begin
   Pos1:= PosY;
   Pos2:= PosX;
   end;
  end;
If (Pos1=0) and (Pos2=0) then goto salto4;
For PosX:= 1 to 100 do
 matrizauxiliar[PosX]:= $ffff;
valormaxim:=0;
```

```
If Pos2= X+1 then
 begin
 For PosX:= 1 to X do
  If Pos1<>PosX then
   If GridSoma2.cells[PosX,Pos1]<>'X' then
   matrizauxiliar[PosX]:= strtoint(GridSoma2.cells[PosX,Pos1]);
   valormaxim:= minvalue(matrizauxiliar);
    For PosY:= 1 to X do
     If GridSoma2.cells[PosY, Pos1]<>'X' then
      begin
       gridBug:= strtofloat(GridSoma2.cells[PosY, Pos1]);
       If gridbug= valormaxim then
         begin
         matrizsolfinalsimmons[PonteiroX-1,0]:= (Pos1);
         matrizsolfinalsimmons[PonteiroX-1,1]:= (PosY);
         PosD:= PosY;
         gridsoma2.cells[Pos1, PosY]:= 'X';
           For Pontei:= 1 to X+1 do
            gridsoma2.cells[PosY, Pontei]:= 'X';
           For.Pontei:= 1 to X+1 do
            gridsoma2.cells[Pontei, Pos1]:= 'X';
         If PosD<>0 then goto salto4;
         end;
   end;
   end
else
 For PosX:= 1 to X do
   If PosX<>Pos2 then
   If GridSoma2.cells[Pos2,PosX]<>'X' then
   matrizauxiliar[PosX]:= strtoint(gridsoma2.cells[Pos2,PosX]);
   valormaxim:= minvalue(matrizauxiliar);
     For PosB:= 1 to X do
      If GridSoma2.cells[Pos2, PosB]<>'X' then
      begin
        gridbug:= strtofloat(GridSoma2.cells[Pos2, PosB]);
```

```
        If gridbug= valormaxim then
        begin
        matrizsolfinalsimmons[PonteiroX-1,0]:= (PosB);
        matrizsolfinalsimmons[PonteiroX-1,1]:= (Pos2);
        PosA:=PosB;
        gridsoma2.cells[PosB, Pos2]:= 'X';
          For Pontei:= 1 to X+1 do
            gridsoma2.cells[Pos2, Pontei]:= 'X';
          For Pontei:= 1 to X+1 do
            gridsoma2.cells[Pontei, PosB]:= 'X';
      If PosA<>0 then goto salto4;
      end;
   end;
end;
For PosX:=0 to 100 do
matrizauxiliar[PosX]:=0;
matrizauxiliar[1]:= matrizsolfinalsimmons[X-1, 1];
matrizauxiliar[X]:= matrizsolfinalsimmons[X-1, 0];
For PosY:= 1 to X-2 do
  For PosX:= 1 to X-1 do
    begin
    If (matrizsolfinalsimmons[PosX-1, 0]) = (matrizauxiliar[PosY]) then
      matrizauxiliar[PosY+1]:= matrizsolfinalsimmons[PosX-1, 1];
    end;
For PosX:= 1 to X do
  begin
  resultsimmons:= resultsimmons + 'J'+ floattostr(matrizauxiliar[PosX]) + ', ';
  end;
EdSeq2.text:= resultsimmons;
For PosX:= 1 to X do
   solucaomelhor[PosX-1]:= strtoint(floattostr(matrizauxiliar[PosX]));
TempoV:= now;
ProResul:= TempoV - TempoJ;
EdTempo2.Text:= FormatDateTime ('h"h"m"m"s"s"zzz"ms"', (ProResul));
end;
```

```
procedure TForm1.calculosgrafico;
var
linha, coluna, PonteiroS, PonteiroJ, PosX, PosY: integer;
tempoespera: real;
begin
For linha:= 0 to Y-1 do
begin
coluna:=0;
ponteiroS:=-1;
ponteiroJ:=0;
 Repeat
  If coluna mod 9=0 then
   begin
    If coluna<9 then
    matrizgrafico[linha, coluna]:= 0
   else
    matrizgrafico[linha, coluna]:=
    strtofloat (matriz[linha, (solucaomelhor[PonteiroS])-1,
    (solucaomelhor[PonteiroS+1])-1]);
   PonteiroS:=PonteiroS+1;
   end;
 coluna:= coluna+1;
 If coluna mod 9=1 then
  begin
   matrizgrafico[linha, coluna]:=
   strtofloat(grid1.cells[(solucaomelhor[PonteiroJ]), linha+1]);
   PonteiroJ:= PonteiroJ+1;
  end;
  Coluna:= coluna+1;
  If coluna mod 9=2 then
   If (linha=0) or (coluna<9) then
    matrizgrafico[linha, coluna]:=0
   else
   begin
   tempoespera:= (matrizgrafico[linha-1, coluna+1]) -
```

```
((matrizgrafico[linha, coluna-8]) + (matrizgrafico[linha, coluna-2]));
 if tempoespera<0 then
  matrizgrafico[linha, coluna]:=0
 else
  matrizgrafico[linha, coluna]:= tempoespera;
end;
Coluna:= coluna+1;
If coluna mod 9=3 then
If (coluna<9) and (linha=0) then
matrizgrafico[linha,coluna]:= 0 +
matrizgrafico[linha, (Coluna -2)]+ matrizgrafico[linha, (coluna-1)] +
matrizgrafico[linha, coluna-3]
else if (coluna<9) and (linha>0) then
matrizgrafico[linha,coluna]:= matrizgrafico[linha-1, (coluna)] +
matrizgrafico[linha, (Coluna -2)]+ matrizgrafico[linha, (coluna-1)] +
matrizgrafico[linha, coluna-3]
else
matrizgrafico[linha,coluna]:= matrizgrafico[linha, coluna-9] +
matrizgrafico[linha, (Coluna -2)]+ matrizgrafico[linha, (coluna-1)] +
matrizgrafico[linha, coluna-3];
coluna:= coluna+1;
If coluna mod 9=4 then
 If coluna<9 then
 matrizgrafico[linha, coluna]:= 0
 else
 matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -10)];
coluna:= coluna+1;
If coluna mod 9=5 then
 matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -5)] +
 matrizgrafico[linha, (Coluna -1)];
coluna:= coluna+1;
If coluna mod 9=6 then
 If (linha>0) and (coluna<9) then
 matrizgrafico[linha, coluna]:= matrizgrafico[linha-1, (coluna-3)]
 else
```

```
    matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -1)]+
    matrizgrafico[linha, (coluna-4)];
 coluna:= coluna+1;
  If coluna mod 9=7 then
    If (linha=0) or (coluna <9) then
      matrizgrafico[linha, coluna]:= 0
    else
      matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -2)];
 coluna:= coluna+1;
  If coluna mod 9=8 then
    matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -1)]+
    matrizgrafico[linha, (coluna-6)];
 coluna:= coluna+1;
until coluna=(9*X);
end;
For linha:= 0 to Y-1 do
 matrizgrafico[linha, (9*X)]:= matrizgrafico[linha, ((9*X)-6)] -
 matrizgrafico[linha, 6];
For PosX:= 1 to 200 do estatica[PosX]:=0;
 For PosY:= 0 to (Y-1) do
  estatica[PosY+1]:= matrizgrafico[PosY, (9*X)];
tempogargalo:= maxValue(estatica);
 For PosX:= 0 to Y-1 do
  If matrizgrafico[PosX,(9*X)]= tempogargalo then gargalo:= PosX+1;
end;

procedure TForm1.calculosgrafico1;
var
linha, coluna, PonteiroS,PonteiroJ: integer;
tempoespera: real;
begin
For linha:= 0 to Y-1 do
begin
coluna:=0;
ponteiroS:=-1;
```

```
ponteiroJ:=0;
 Repeat
  If coluna mod 9=0 then
   begin
    If coluna<9 then
    matrizgrafico[linha, coluna]:= 0
   else
    matrizgrafico[linha, coluna]:=
    strtofloat (matriz[linha, (solucaomelhor[PonteiroS])-1,
    (solucaomelhor[PonteiroS+1])-1]);
   PonteiroS:=PonteiroS+1;
   end;
 coluna:= coluna+1;
 If coluna mod 9=1 then
  begin
   matrizgrafico[linha, coluna]:=
   strtofloat(grid1.cells[(solucaomelhor[PonteiroJ]), linha+1]);
   PonteiroJ:= PonteiroJ+1;
  end;
  Coluna:= coluna+1;
  If coluna mod 9=2 then
   If (linha=0) or (coluna<9) then
    matrizgrafico[linha, coluna]:=0
   else
   begin
   tempoespera:= (matrizgrafico[linha-1, coluna+1]) -
   ((matrizgrafico[linha, coluna-8]) + (matrizgrafico[linha, coluna-2]));
    if tempoespera<0 then
     matrizgrafico[linha, coluna]:=0
    else
     matrizgrafico[linha, coluna]:= tempoespera;
   end;
   Coluna:= coluna+1;
   If coluna mod 9=3 then
   If (coluna<9) and (linha=0) then
```

```
   matrizgrafico[linha,coluna]:= 0 +
   matrizgrafico[linha, (Coluna -2)]+ matrizgrafico[linha, (coluna-1)] +
   matrizgrafico[linha, coluna-3]
   else if (coluna<9) and (linha>0) then
   matrizgrafico[linha,coluna]:= matrizgrafico[linha-1, (coluna)] +
   matrizgrafico[linha, (Coluna -2)]+ matrizgrafico[linha, (coluna-1)] +
   matrizgrafico[linha, coluna-3]
  else
   matrizgrafico[linha,coluna]:= matrizgrafico[linha, coluna-9] +
   matrizgrafico[linha, (Coluna -2)]+ matrizgrafico[linha, (coluna-1)] +
   matrizgrafico[linha, coluna-3];
coluna:= coluna+1;
 If coluna mod 9=4 then
   If coluna<9 then
   matrizgrafico[linha, coluna]:= 0
   else
   matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -10)];
coluna:= coluna+1;
  If coluna mod 9=5 then
   matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -5)] +
   matrizgrafico[linha, (Coluna -1)];
coluna:= coluna+1;
  If coluna mod 9=6 then
   If (linha>0) and (coluna<9) then
   matrizgrafico[linha, coluna]:= matrizgrafico[linha-1, (coluna-3)]
   else
   matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -1)]+
   matrizgrafico[linha, (coluna-4)];
coluna:= coluna+1;
  If coluna mod 9=7 then
   If (linha=0) or (coluna <9) then
    matrizgrafico[linha, coluna]:= 0
   else
    matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -2)];
coluna:= coluna+1;
```

```
  If coluna mod 9=8 then
    matrizgrafico[linha,coluna]:= matrizgrafico[linha, (Coluna -1)]+
    matrizgrafico[linha, (coluna-6)];
 coluna:= coluna+1;
until coluna=(9*X);
end;
EdMak2.Text:= floattostr (matrizgrafico[(Y-1), ((9*X)-6)]);
end;

procedure TForm1.ParmetrosSA1Click(Sender: TObject);
var
linha1, coluna1, maquina1, PosX, PosY: integer;
espacos: string;
begin
memo1.lines.Clear;
linha1:=0;
memo1.lines.add( '*.*.*. RELATÓRIO DE RESOLUÇÃO DE PROBLEMA .*.*.* ');
memo1.lines.add('');
memo1.lines.add('DADOS DO PROBLEMA');
memo1.lines.add( 'No. de máquinas: ' + inttostr (Y));
memo1.lines.add( 'No. de tarefas: ' +  inttostr (X));
memo1.lines.add( 'Range Tempos de Processamento: ' +  EdTp.Text);
memo1.lines.add('Range Tempos de Setup: ' +  EdTxij.Text);
memo1.lines.add('');
memo1.lines.add( 'PARÂMETROS SIMULATED ANNEALING');
memo1.lines.add( 'Número de Iterações: ' + LbIteracoes.Text);
memo1.lines.add( 'Temperatura Inicial: ' + LblTempInicial.Text);
memo1.lines.add( 'Temperatura Final: ' + LblTempFinal.Text);
memo1.lines.add( '');
INC(linha1,12);
memo1.lines.add( 'MATRIZ DE TEMPOS DE PROCESSAMENTO');
Linha1:= linha1 + 2;
memo1.lines.add( '  |');
For coluna1:= 1 to X do
  memo1.lines[linha1]:= memo1.lines[linha1] + ' J' + Formatfloat('00',coluna1)
```

```
        + '|';
  Linha1:= linha1 + 1;
   For maquina1:= 1 to Y do
   begin
    memo1.lines.add('M' + Formatfloat ('00', maquina1) + '|');
     For coluna1:= 1 to X do
     begin
     Case strtoint(grid1.Cells [coluna1, maquina1]) of
      0..9: espacos := '  ';
      10..99: espacos := ' ';
      100..999: espacos := ' ';
     else
      espacos:= '';
     end;
    memo1.lines[linha1]:= memo1.lines[linha1]  + espacos + Grid1.Cells [coluna1,
maquina1]
      + '|';
   end;
   Linha1:= linha1+1;
   end;
   memo1.lines.add( '');
Linha1:= linha1 +1 ;
PosY:= roller.Position:
memo1.lines.add( 'MATRIZES DE TEMPOS DE SETUP');
Linha1:= linha1 +1 ;
roller.Position:= 0;
For PosX:= 1 to Y do
 begin
 memo1.lines.Add('M'+Formatfloat ('00', PosX)+'|');
  For coluna1:= 1 to X do
   memo1.lines[linha1]:= memo1.lines[linha1] + ' J' + Formatfloat('00',coluna1)
       + '|';
   Linha1:= linha1 + 1;
   For maquina1:= 1 to X do
   begin
```

```
      memo1.lines.Add( 'J' + Formatfloat ('00', maquina1) + '|');
       For coluna1:= 1 to X do
       begin
        If coluna1=maquina1 then memo1.lines[linha1]:= memo1.lines[linha1] + ' ';
        If coluna1<>maquina1 then
         begin
         Case strtoint(grid2.Cells [coluna1, maquina1]) of
          0..9: espacos := '  ';
          10..99: espacos := ' ';
          100..999: espacos := ' ';
         else
          espacos:= '';
        end;
        end;
        memo1.lines[linha1]:= memo1.lines[linha1] + espacos + Grid2.Cells [coluna1,
maquina1]
                + '|';
         end;
         linha1:= linha1+1;
          end;
roller.Position:= roller.Position+1;
INC (linha1,1);
memo1.Lines.Add('');
end;
memo1.Lines.add ('RESULTADOS BGaFSA');
memo1.Lines.add ('Gargalo Inicial: ' + EdGargaloInicial.text);
memo1.Lines.add ('Makespan: ' + Mak2.Text);
memo1.Lines.add ('Gargalo Final: ' + EdMaqGargalo.text);
memo1.Lines.add ('Makespan: ' + Edit1.Text);
memo1.Lines.add ('Melhora Relativa ' + Lbmelhora.text);
memo1.Lines.add ('Melhor Iteração do S.A.: '+ EdIteracao1.text);
memo1.Lines.add ('Tempo de processamento: '+ EdTempo.Text);
memo1.Lines.add ('Melhor seqüência: '+ Edit2.Text);
memo1.Lines.add ('');
memo1.Lines.add ('RESULTADOS SIMMONS Jr. - TOTAL');
```

```
memo1.Lines.add ('Makespan: ' + EdMak2.Text);
memo1.Lines.add ('Tempo de processamento: '+ EdTempo2.Text);
memo1.Lines.add ('Melhor seqüência: '+ EdSeq2.Text);
memo1.Lines.add ('');
memo1.Lines.add ('COMPARATIVO ENTRE OS MÉTODOS');
memo1.Lines.add ('Melhor Método: '+ LabeledEdit1.Text);
memo1.Lines.add ('Melhora Relativa Percentual: ' + LabeledEdit2.Text);
roller.Position:= PosY;
memo1.Print('Relatório');
end;

procedure TForm1.Imprimirmatrizes1Click(Sender: TObject);
begin
memo1.lines.Clear;
memo1.lines.add( '*.*.*. RELATÓRIO DE RESOLUÇÃO DE PROBLEMA .*.*.* ');
memo1.lines.add('');
memo1.lines.add('DADOS DO PROBLEMA');
memo1.lines.add( 'No. de máquinas: ' + inttostr (Y));
memo1.lines.add( 'No. de tarefas: ' +  inttostr (X));
memo1.lines.add( 'Range Tempos de Processamento: ' +  EdTp.Text);
memo1.lines.add('Range Tempos de Setup: ' +  EdTxij.Text);
memo1.lines.add('');
memo1.lines.add( 'PARÂMETROS SIMULATED ANNEALING');
memo1.lines.add( 'Número de Iterações: ' + LbIteracoes.Text);
memo1.lines.add( 'Temperatura Inicial: ' + LblTempInicial.Text);
memo1.lines.add( 'Temperatura Final: ' + LblTempFinal.Text);
memo1.lines.add( '');
memo1.Lines.add ('RESULTADOS BGaFSA');
memo1.Lines.add ('Gargalo Inicial: ' + EdGargaloInicial.text);
memo1.Lines.add ('Makespan: ' + Mak2.Text);
memo1.Lines.add ('Gargalo Final: ' + EdMaqGargalo.text);
memo1.Lines.add ('Makespan: ' + Edit1.Text);
memo1.Lines.add ('Melhora Relativa ' + Lbmelhora.text);
memo1.Lines.add ('Melhor Iteração do S.A.: '+ EdIteracao1.text);
memo1.Lines.add ('Tempo de processamento: '+ EdTempo.Text);
```

```
memo1.Lines.add ('Melhor seqüência: '+ Edit2.Text);
memo1.Lines.add ('');
memo1.Lines.add ('RESULTADOS SIMMONS Jr. - TOTAL');
memo1.Lines.add ('Makespan: ' + EdMak2.Text);
memo1.Lines.add ('Tempo de processamento: '+ EdTempo2.Text);
memo1.Lines.add ('Melhor seqüência: '+ EdSeq2.Text);
memo1.Lines.add ('');
memo1.Lines.add ('COMPARATIVO ENTRE OS MÉTODOS');
memo1.Lines.add ('Melhor Método: '+ LabeledEdit1.Text);
memo1.Lines.add ('Melhora Relativa Percentual: ' + LabeledEdit2.Text);
memo1.Print('Relatório');;
 end;

procedure TForm1.Limparresultados1Click(Sender: TObject);
var
Indice, PosX: integer;
begin
 for Indice:=1 to X+1 Do
  begin
  Grid3.cols[indice].clear;
  Grid4.Cols[indice].clear;
  Grid5.Cols[indice].clear;
  GridInv.Cols[Indice].clear;
  GridSoma2.cols[indice].clear;
  end;
  List1.Items.clear;
  Edit1.Text:= '';
  Edit2.Text:= '';
  EdIteracao1.Text:= '';
  EdMak2.Text:= '';
  EdMaqGargalo.Text:= '';
  EdSeq2.Text:= '';
  EdTempo.Text:= '';
  EdTempo2.Text:= '';
  LabeledEdit1.Text:= '';
```

```
  LabeledEdit2.Text:= '';
  LbIteracoes.Text:= '';
  LblTempfinal.Text:= '';
  LblTempInicial.text:= '';
  EdGargaloInicial.text:= '';
  Mak2.Text:='';
  resultsimmons:= '';
  Iteracoes1:=0;
  ItTotal:=0;
  gargalo:=0;
  linhaMaior:=0;
  tempoinicial:=0;
  tempotemporario:=0;
  tempomelhor:=0;
  TIni:=0;
  TFinal:=0;
  TempCorr:=0;
  tempogargalo:=0;
  bestmakespan:=0;
  {For PosX:= 0 to X do  solucaoInicial[PosX]:=0;
  For PosX:= 0 to X do  solucaoFinal[PosX]:=0;
  For PosX:= 0 to X do  solucaoTemp[PosX]:=0;
  For PosX:= 0 to X do  solucaoMelhor[PosX]:=0;
  For PosX:= 0 to X-1 do  calculo[PosX]:=0;
  For PosX:= 0 to Y do  matrizgrafico[PosX]:=0;
  //For PosX:= 0 to ((X-1)*(X-1)) do  matrizinsertion[PosX]:=0;
  //For PosX:= 0 to Y do  matrizsolfinaldetecta[PosX]:=0;
  For PosX:= 0 to 100 do  matrizsolfinalsimmons[PosX]:=0;
  For PosX:= 0 to Y do  matrizsolfinalmakespan[PosX]:=0;
  For PosX:= 0 to Y do  matrizsolfinaltours[PosX]:=0;  }
end;

procedure TForm1.Montargrafico1Click(Sender: TObject);
begin
Grafico.show;
```

```
end;

procedure TForm1.Imprimirgrfico1Click(Sender: TObject);
begin
Grafico.bitbtn3.click;
end;

end.



unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, TeEngine, Series, GanttCh, ExtCtrls, TeeProcs, Chart, StdCtrls,
  Buttons;

type
  TGRAFICO = class(TForm)
    Chart1: TChart;
    Series1: TGanttSeries;
    BitBtn1: TBitBtn;
    Chart3: TChart;
    GanttSeries2: TGanttSeries;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
```

```
 public
  { Public declarations }
 end;

var
 GRAFICO: TGRAFICO;

implementation

uses Unit1;

{$R *.dfm}

procedure TGRAFICO.BitBtn1Click(Sender: TObject);
var
XIni, XEnd: double;
Nome: string;
PosX, PosY: integer;
Label
SALTOX;
begin
series1.clear;
For PosY:= 0 to Y-1 do
Begin
PosX:=1;
Repeat
XIni:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome:= 'J' + inttostr(solucaomelhor[PosX-1]);
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clteal);
If PosX=X then goto SALTOX;
INC (PosX);
XIni:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome:= 'J' + inttostr(solucaomelhor[PosX-1]);
```

```
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clYellow);
If PosX=X then goto SALTOX;
INC (PosX);
XIni:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome:= 'J' + inttostr(solucaomelhor[PosX-1]);
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clFuchsia);
If PosX=X then goto SALTOX;
INC (PosX);
XIni:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome:= 'J' + inttostr(solucaomelhor[PosX-1]);
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clAqua);
If PosX=X then goto SALTOX;
INC (PosX);
XIni:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome:= 'J' + inttostr(solucaomelhor[PosX-1]);
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clRed);
SALTOX:
Until PosX>=X;
end;
For PosY:= 0 to Y-1 do
Begin
PosX:=2;
Repeat
XIni:= matrizGrafico [PosY, (4+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (5+((PosX-1)*9))];
Nome:= 'S';
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clmoneygreen);
INC (PosX);
Until PosX>X;
end;
For PosY:= 0 to Y-1 do
Begin
```

```
PosX:=2;
Repeat
XIni:= matrizGrafico [PosY, (7+((PosX-1)*9))];
XEnd:= MatrizGrafico [PosY, (8+((PosX-1)*9))];
Nome:= 'E';
If Xini<>XEnd then
Series1.AddGanttColor(XIni, XEnd, PosY+1, Nome, clWhite);
INC (PosX);
Until PosX>X;
end;
end;

procedure TGRAFICO.BitBtn2Click(Sender: TObject);
var
XIni1, XEnd1: double;
Nome1: string;
PosX, PosY: integer;
Label
SALTOXX;
begin
GanttSeries2.clear;
For PosY:= 0 to Y-1 do
Begin
PosX:=1;
Repeat
XIni1:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome1:= 'J' + inttostr(solucaomelhor[PosX-1]);
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clteal);
If PosX=X then goto SALTOXX;
INC (PosX);
XIni1:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome1:= 'J' + inttostr(solucaomelhor[PosX-1]);
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clYellow);
```

```
If PosX=X then goto SALTOXX;
INC (PosX);
XIni1:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome1:= 'J' + inttostr(solucaomelhor[PosX-1]);
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clFuchsia);
If PosX=X then goto SALTOXX;
INC (PosX);
XIni1:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome1:= 'J' + inttostr(solucaomelhor[PosX-1]);
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clAqua);
If PosX=X then goto SALTOXX;
INC (PosX);
XIni1:= matrizGrafico [PosY, (6+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (3+((PosX-1)*9))];
Nome1:= 'J' + inttostr(solucaomelhor[PosX-1]);
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clRed);
SALTOXX:
Until PosX>=X;
end;
For PosY:= 0 to Y-1 do
Begin
PosX:=2;
Repeat
XIni1:= matrizGrafico [PosY, (4+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (5+((PosX-1)*9))];
Nome1:= 'S';
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clmoneygreen);
INC (PosX);
Until PosX>X;
end;
For PosY:= 0 to Y-1 do
Begin
PosX:=2;
```

```
Repeat
XIni1:= matrizGrafico [PosY, (7+((PosX-1)*9))];
XEnd1:= MatrizGrafico [PosY, (8+((PosX-1)*9))];
Nome1:= 'E';
If Xini1<>XEnd1 then
GanttSeries2.AddGanttColor(XIni1, XEnd1, PosY+1, Nome1, clWhite);
INC (PosX);
Until PosX>X;
end;
end;

procedure TGRAFICO.BitBtn3Click(Sender: TObject);
begin
print;
end;

end.
```

REFERÊNCIAS BIBLIOGRÁFICAS

ALLAHVERDI, A.; GUPTA, J.N.D.; ALDOWAISAN, T. (1999). A review of scheduling involving setup considerations. *Omega*, v.27, p.219-239.

ARAÚJO, J.S. (1971). *Administração de materiais.* 2 ed. São Paulo, Atlas.

BAKER, K.R. (1974). *Introduction to Sequencing and Scheduling.* New York, John Wiley and Sons. Cap. 6, p.136-176: Flow Shop scheduling.

BLACK, J.T. (1983). Cellular manufacturing systems reduce setup time, make small lot production economical. *Industrial Engineering*, v.15, n.11, p.36-48.

BUZZO, W.R.; MOCCELLIN, J.V.(1999). A influência da temperatura inicial no desempenho de um método híbrido algoritmo genético – simulated annealing para a programação flow shop permutacional. Encontro Nacional de Engenharia de Produção, 5/ Internatinal Congress of Industrial Engineering, 3. *Proceedings*, Rio de Janeiro.

CONWAY, R.W.; MAXWELL, W.L.; MILLER, L.W. (1976). *Theory of Scheduling.* New York, Addison-Wesley. Cap. 5, p.80-102: Flow Shop scheduling.

DAS, S.R.; GUPTA, J.N.D.; KHUMAWALA, B.M. (1995). A savings index heuristic algorithm for flow shop scheduling with sequence dependent set-up times. *Journal of the Operational Research Society*. v.46, p.1365-1373.

FISCHETTI, M; TOTH, P. (1997). A polyhedral approach to the asymmetric Traveling Salesman Problem. *Management Science*, v.43, n.11, p.1520-1536, november.

FLYNN, B.B. (1987). The effects of setup time on output capacity in cellular manufacturing. *International Journal of Operations Research*, v.25, n.12, p.1761-1772.

FLOOD, M.M. (1956). The Traveling-Salesman Problem. *Operations Research* 4, no.1, p.61-75, feb..

FOGARTY, D.W.; BLACKSTONE JR, J.H.; HOFFMANN, T.R. (1983). *Production and Inventory Management*, 2.ed. Ohio, South-Western.

GARFINKEL, R.S. (1973). On partitioning the feasible set in a Branch-and-Bound algorithm for the asymmetric Traveling Salesman Problem. *Operations Research*, v.21, p.341-345.

GAVETT, J.W. (1965). Three heuristic rules for sequencing jobs to a single production facility. *Management Science*, v.11, n.8, p.B166-B176.

GOLDRATT, E.; FOX, J. (1997). *A meta: um processo de aprimoramento contínuo.* São Paulo. Educador.

HAX, A.C.; CANDEA, D. (1984). *Production and Inventory Control.* New Jersey, Prentice Hall. Cap.5, p.257-392: Operations Scheduling.

KANELLAKIS, P.C.; PAPADIMITRIOU, C.H. (1980). Local search for the asymmetric Traveling Salesman Problem. *Operations Research*, v.28, n.5, p.1086-1099, october.

KARMARKAR, U.S. (1987). Lot Sizes, Lead Times and In-process Inventories. *Management Science,* v.33, n.3, p.409-418, march.

KIM, S.C.; BOBROWSKI, P.M. (1994). Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Operations Research*, v.32, n.7, p.1503-1520.

KIRKPATRICK, S.; GELATT JR, C.D.; VECCHI, M.P. (1983). Optimization by Simulated Annealing, *Science*, v.220, n.4598, p.671-680, may.

LITTLE, J.D.C.; MURTY, K.G.; SWEENEY, D.W; KAREL, C. (1963). An algorithm for the Traveling-Salesman Problem. *Operations Research*, v.11, n.6, p.972-989, nov..

LOCKETT, A.G.; MUHLEMANN, A.P.(1972). A scheduling problem involving sequence dependent changeover times. *Operations Research*, v.20, p.895-902.

MACCARTHY, B.L.; LIU, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, v.31, n.1, p.59-79.

MOCCELLIN, J.V. (1994). Comparsion of neighborhood search heuristics for flowshop sequencing problem. Fourth International Workshop on Project Management and Scheduling, July. Leuven – Belgium. *Proceedings*, p.228-231.

OSMAN, I.H.; POTTS, C.N. (1989). Simulated Annealing for permutation flow-shop scheduling. *Omega*, v.17, n.6, p.551-557.

PARTHASARATHY, S.; RAJENDRAN, C. (1997). A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flow shop with sequence-dependent setup times of jobs - a case study. *Production Planning & Control*, v.8, n.5, p.475-483.

PATTERSON, M.C. (1993). Analysis of setup time at constraint resources. *International Journal of Operations Research*, v.31, n.4, p.845-849.

PINEDO, M. (1995). *Scheduling: theory, algorithms and systems.* 1.ed. Prentice-Hall.

PLOSSL, G.W. (1985). *Production and inventory control.* 2.ed. New Jersey, Prentice Hall.

RAJENDRAN, C.; ZIEGLER,H. (1997). A Heuristic for Scheduling to Minimize the Sum of Weighted Flow Time of Jobs in a Flow Shop with Sequence Dependent Setup Times of Jobs. *Computers Industrial Engineers,* v.33, n1-2, p.281-284.

RíOS-MERCADO, R.Z.; BARD, J.F. (1998). Heuristics for the flow line problem with setup costs. *European Journal of Operations Research*, v.110, p.76-98.

RUBIN, P.A.; RAGATZ, G.L. (1995). Scheduling in a sequence dependent setup environment with genetic search. *Computers Operations Research*, v.22, n.1, p.85-99.

SIMONS JR, J.V. (1992). Heuristics in flow shop scheduling with sequence dependent setup times. *Omega*, v.20, n.2, p.215-225.

SLACK, N. et. al. (1996). *Administração da Produção*. São Paulo, Atlas. Cap.10, p.229-252: Natureza do planejamento e controle.

SPENCE, A.M.; PORTEUS, E.L. (1987). Setup reduction and increased effective capacity. *Management Science*, v.33, n.10, p.1291-1301.

STEIN, R.E. (1997). *The Theory of Constraints: applications in quality and manufacturing*. 2.ed. New York. Marcel Dekker. Cap.2, p.5-16, Creating the Process of Continuous Profit Improvement.

TAN, K.C.; NARASIMHAN, R; RUBIN, P.A.; RAGATZ, G.L. (2000). A comparsion of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. *Omega*, v.28, p.313-326.

WILBRECHT, J.K.; PRESCOTT, W.B. (1969). The influence of setup time on job shop performance. *Management Science*, v.16, n.4, p.B274-B280, december.

YANG, J.; DEANE, R.H. (1993). Setup time reduction and competitive advantage in a closed manufacturing cell. *European Journal of Operations Research*, n.69, p.413-423.