UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO
ESCOLA DE ADMINISTRAÇÃO
UNIVATES – CENTRO UNIVERSITÁRIO
MESTRADO EM ADMINISTRAÇÃO

LUCIANO BRANDÃO

SEQUENCIAMENTO DE PROCESSADORES PARALELOS UTILIZANDO A META HEURÍSTICA BUSCA TABU

LAJEADO
2002

LUCIANO BRANDÃO

SEQUENCIAMENTO DE PROCESSADORES PARALELOS UTILIZANDO A META HEURÍSTICA BUSCA TABU

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Administração da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do título de Mestre em Administração.

Orientador: Prof. Dr. Denis Borenstein

LAJEADO

2002

Dados Internacionais de Catalogação na Publicação (CIP)

B817s Brandão, Luciano

Sequenciamento de processadores paralelos utilizando a meta heurística busca tabu/ Luciano Brandão. – 2002.

68 f. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul, Escola de Administração, Programa de Pós-Graduação em Administração, 2002.

"Orientador: Prof. Dr. Dênis Borenstein".

1. Gestão de processos. 2. Administração da produção. 3. Processadores paralelos - Busca TABU I. Título.

CDU 658.5

Ficha elaborada pela equipe da Biblioteca da Escola de Administração UFRGS

ESCOLA DE ADMINISTRAÇÃO
R. Washington Luiz, 855 Prédio 13701 - Campus Centro
Porto Alegre – RS - 90010-460
Fone: 55 51 3308 3536 Fax: 55 51 3308 3991
E-mail: especializacao@ea.ufrgs.br

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO EM ADMINISTRAÇÃO
ESCOLA DE ADMINISTRAÇÃO
UNIVATES – CENTRO UNIVERSITÁRIO
MESTRADO EM ADMINISTRAÇÃO

A banca examinadora, abaixo assinada, aprova a dissertação intitulada "Sequenciamento de processadores paralelos utilizando a meta heurística busca Tabu" elaborada por Luciano Brandão, como requisito parcial para obtenção do grau de Mestre em Administração:

_____
Prof. Dr.João Luiz Becker

_____
Prof Dr. Flávio Fogliatto

_____
Prof Dr. Denis Borenstein

Dedico este trabalho à minha esposa Soraide, meus pais, e todos que de alguma forma contribuíram ou foram afetados pelo desenvolvimento deste trabalho.

Um agradecimento especial a todos que contribuíram para este trabalho, ao meu orientador e amigo Prof. Dr. Denis Borenstein, a todos os professores do PPGA-UFRGS, ao João B. Gravina, pelas nossas agradáveis viagens Lajeado - Porto Alegre - Lajeado, aos colegas que participaram deste processo, ao Prof. Fred Glover pelo material enviado.

Pensamento:

“Gostaria de saber hoje metade do que pensava saber quando tinha 18 anos.”

(Michelangelo aos 80 anos)

RESUMO

A programação de tarefas em linhas de produção nas empresas sempre foi e continua sendo um elemento fundamental para o sucesso das organizações em um mercado tão globalizado e competitivo. A melhor utilização dos recursos instalados através da melhor alocação das tarefas gerará melhores resultados para a organização. Entende-se pela melhor utilização dos recursos a redução do tempo total de finalização das tarefas (*makespan*) sem prejudicar o atendimento da data de entrega. Aplica-se esta idéia para as indústrias de um modo geral, que tenham linhas de produção, podendo citar a indústria calçadista, foco neste trabalho, as indústrias de massas, biscoitos e balas, entre outras. Na literatura especializada, esta programação é conhecida como sequenciamento de tarefas em processadores. Neste trabalho aplicado junto a indústria calçadista, foca-se em uma área mais específica: o sequenciamento de tarefas em processadores paralelos. Os problemas de sequenciamento se caracterizam pela grande exigência computacional para a resolução com algoritmos de otimização. Isto remete a utilização de heurísticas para a resolução destes problemas. Neste trabalho explora-se a Meta-Heurística Busca Tabu, que se apresentou com resultados muito bons em relação ao ótimo e em relação ao trabalhador humano.

Palavras chave: Gestão de Processos. Administração da Produção. Processadores Paralelos - Busca TABU.

ABSTRACT

The jobs scheduling in processor lines in the companies was always, and continues being, a fundamental element to the organization's success in a very globalized and competitive market. The best use of the installed resources, through the best distribution of the jobs will generate better results for the organization. The best utilization of the resources means the reduction of the makespan without prejudicing the due-date. This idea is applied to all industries in general, that have processor lines, for example the shoes factories, focused in this research, pasta, cookies and sugar balls factories, beyond others. In the specialized literature this subject is know as job scheduling. This researh is applyed to a shoes factory is focused on more specific area: the job scheduling in parallel machines. The scheduling problems are characterized on its computational difficulty using optimization algoritms. That is the reason why we used heuristics to solve these problems. In this research we explore the Meta-Heuristic Tabu Search, wich showed very good results comparing to the optimun and comparing to the human worker.

Keywords: Process Management. Production Management. Parallel Processors - TABU Search.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O problema de sequenciamento de tarefas em processadores paralelos recebe muita atenção dos pesquisadores, vide Weng, Lu e Ren (2001); Chen (1995); Müller (1993); Wathier (1999); Cheng (1989); Moccellin e Nagano (1998); Nowicki e Smutnicki (1998); Adamopoulos e Pappis (1998); Drees e Wilhelm (2001); Dessouky, Dessouky e Verma (1998) entre outros. Este problema tem aplicações em várias organizações que tenham linhas de produção, podendo citar a indústria calçadista, foco neste trabalho, as indústrias de massas, biscoitos e balas, entre outras. Parte destes estudos é destinado para uma categoria específica de sequenciamento, com tempo de *setup* dependente da seqüência das tarefas.

Na literatura, incluindo a citada anteriormente, encontramos estudos a respeito de máquinas semelhantes e processadores paralelos independentes. Outros trabalhos fazem referência a processadores paralelos independentes e com tempo de *setup* dependente da seqüência das tarefas, com o objetivo de reduzir o *makespan*, ou seja, o tempo de processamento de todos os processadores.

A complexidade destes algoritmos desenvolvidos impede sua aplicação prática para problemas muito grandes. Desta forma é necessário o desenvolvimento de heurísticas mais eficientes. Este trabalho utiliza a meta heurística Busca Tabu para a resolução de problemas de processadores paralelos, com o objetivo de reduzir o atraso geral das tarefas ou a redução do *makespan*. A Busca Tabu, desenvolvida por Fred Glover, vem sendo utilizada com sucesso por diversos autores como: Moccellin (1995); Müller (1993); Wathier (1999); Negenman (2001); e, Nagar, Heragu e Haddock (1995), para a resolução de problemas de escalonamento em processadores.

Ao final deste trabalho são apresentados os resultados alcançados após a aplicação de um algoritmo desenvolvido utilizando a Busca Tabu. Esta aplicação está inserida em um ambiente de produção, com vários processadores, que tem o escalonamento feito por pessoas de forma manual. Atualmente, novas solicitações de produção necessitam, para sua aprovação, de disponibilidade nos processadores. O objetivo deste trabalho é reduzir o *makespan* sem prejudicar a entrega das tarefas e, consequentemente, conseguir atender mais pedidos de produção. Várias situações foram analisadas, a seguir:

a) comparação dos resultados obtidos com o algoritmo desenvolvido com os resultados obtidos com a programação linear inteira;
b) redução do *makespan*, sem comprometer as datas de entrega das tarefas.

Após as análises, pode-se constatar que o algoritmo se mostra eficiente e muito mais rápido do que as soluções de otimização utilizando Programação Linear Inteira. Em comparação com escalonadores humanos o algoritmo conseguiu reduzir o *makespan*, em média, 27%, com um tempo de processamento máximo de 5 minutos nos testes aplicados, ao passo que, manualmente, o serviço consome quase um dia inteiro e o envolvimento de mais de uma pessoa.

Este documento está organizado como segue: primeiro aprofunda-se os conceitos de sequenciamento de tarefas em processadores paralelos; em seguida explora-se o entendimento de heurísticas e o porquê das heurísticas serem importantes e necessárias para a resolução deste tipo de problema. Após, apresenta-se a meta-heurística Busca Tabu, desenvolvida por Fred Glover na década de 80. Em seguida, apresenta-se o objetivo geral, objetivos específicos do trabalho e a metodologia aplicada para a resolução do problema. A modelagem, logo a seguir, apresenta uma adaptação do algoritmo proposto por Müller (1993). Em um capítulo a parte, a validação apresenta um modelo matemático de otimização, utilizando Programação Linear Inteira. Também mostra o seu desenvolvimento, a comparação dos resultados obtidos com este modelo matemático e os resultados obtidos com a aplicação do algoritmo apresentado, aplicando a Busca Tabu, utilizando o mesmo conjunto de tarefas e processadores. Para encerrar, são feitas avaliações dos resultados obtidos pelo algoritmo em comparação com dados históricos recolhidos junto a Indústria de Calçados Blip Ltda., empresa do setor calçadista.

2 O SEQUENCIAMENTO DE TAREFAS EM PROCESSADORES PARALELOS

O problema apresentado por este estudo consiste em alocar n tarefas a m processadores idênticos com o objetivo de minimizar o tempo máximo de finalização (*makespan*), conhecido como o Problema de Sequenciamento em Processadores Paralelos. Pode ocorrer que, para processar uma tarefa distinta da tarefa anterior num processador, seja necessário realizar operações de reparo do processador para receber e executar a próxima tarefa. Como exemplo destas operações de preparação do processador estão: limpeza, troca de peças e ferramentas, aquecimento ou resfriamento, ajustes no leiaute do processador e ajustes de um modo em geral. Estas operações feitas no processador consomem um tempo pré-determinado, denominado tempo de preparação (*setup time*).

A partir de um conjunto de processadores idênticos que, por hipótese, têm o mesmo estado inicial, os tempos de processamento total envolvem a soma de duas parcelas: o tempo de preparação do processador para a troca da tarefa i para j, e o tempo de processamento da tarefa j, propriamente dito. Portanto, o problema não é simplesmente alocar as tarefas nos processadores, mas também, encontrar a melhor seqüência de execução das mesmas de modo a minimizar o tempo de execução total. Este problema é conhecido, conforme Pinedo (1995) como problema de sequenciamento em processadores paralelos com tempos de processamento dependentes da seqüência – $P \mid tdps \mid C\max$.

O $P \mid tdps \mid C\max$ é um Problema da Otimização Combinatorial (POC). Um problema de otimização combinatorial procura encontrar uma solução que minimiza (ou maximiza) uma função objetivo num conjunto combinatorial (discreto) de soluções viáveis. O $P \mid tdps \mid C\max$ busca minimizar o tempo máximo de finalização de todas as tarefas.

Uma descrição mais detalhada do problema apresenta a necessidade de alocar *n* tarefas $J_1, J_2, ..., J_3$, a m processadores paralelos idênticos, $M_1, M_2, ..., M_m$, de uma forma não-preemptiva, isto é, uma vez que determinada tarefa esteja alocada a um processador, ela deverá permanecer no mesmo até o seu término, sem interrupções, objetivando minimizar o tempo máximo de finalização de todas as tarefas (*makespan*).

O tempo de processamento de uma tarefa J_j, é composto da soma de duas parcelas: um tempo de execução inteiro e positivo p_j e um tempo de preparação inteiro e positivo para a tarefa J_j, se ela foi executada imediatamente após a tarefa i, chamada s_{ij}. Portanto, o tempo de processamento pode ser expresso como $t_{ij} = p_j + s_{ij}$. Assume-se que não existe nenhuma relação de simetria, isto é, $t_{ij} \neq t_{ji}$.

Este problema pode-se ser aplicado em várias indústrias, que entre o final da produção de um determinado item, e o início de outro, uma ou várias máquinas tem que ser ajustadas, ou o leiaute tem que ser mudado, ou alguma outra configuração da planta industrial tem que ser alterada para que o novo produto possa ser produzido. Isto se aplica em diversos setores industriais, onde os produtos produzidos são vários na mesma linha de produção, podendo citar a indústria calçadista, de tintas, de alimentos, de móveis em série, etc.

Conforme afirma Gerasoulis e Yang (1997), do ponto de vista teórico, todos os problemas de sequenciamento em processadores paralelos são extremamente difíceis no sentido de encontrar uma solução ótima. Então podemos classificá-lo como NP, ou seja, para problemas de sequenciamento pequenos podemos utilizar métodos de otimização (LENSTRA; SHMOYS, 1997), mas quando o problema aumenta em número de tarefas e processadores, precisamos utilizar heurísticas, que nos orientam a uma solução próxima da ótima. Segundo Müller (1993), heurísticas são critérios ou métodos computacionais para decidir o caminho mais eficiente entre várias alternativas de ação, buscando encontrar um determinado objetivo, porém não necessitam encontrar o melhor caminho entre todas as alternativas possíveis, justificando assim a aplicação de uma heurística neste trabalho.

A vantagem no uso de heurísticas está no fato de que elas aumentam a eficiência do processo de busca retornando uma solução boa (que muitas vezes pode ser ótima), em detrimento da exploração de todas as alternativas. Uma motivação para seu uso é que a solução ótima do problema raramente é requerida e as heurísticas raramente apresentam complexidade superior a solução algorítmica ótima.

3 HEURÍSTICA

De acordo com Holanda (1986), o termo “heurística” refere-se a uma abordagem utilizada a um conjunto com diversas regras e métodos que conduzem à descoberta, à invenção e a resolução de problemas. Esta expressão tem vários significados, mas apresenta-se neste trabalho como uma abordagem que não irá garantir que os algoritmos propostos chegarão aos melhores resultados, ou seja, resultados ótimos. Esta é a grande diferença para a programação linear (PL), que leva o algoritmo proposto à resolução do problema para a solução ótima. A programação linear é caracterizada por ser um método de otimização de resultados.

De acordo com Pidd (1998), as abordagens heurísticas, assim como a simulação, não nos garantem ótimas soluções para os problemas propostos. A característica que diferencia estas duas abordagens parte da interação do modelador. A simulação é baseada em experimentos conduzidos por pessoas. A idéia é que o modelador pense sobre possíveis soluções e aplique as possíveis alternativas no modelo. Esta abordagem é conhecida como “*what if*” (o que aconteceria se ...?). As abordagens heurísticas são baseadas em aproximações do resultado ótimo, de forma automática, aos problemas formulados. Não há interação do modelador na busca da melhor solução. É a inteligência e a capacidade do algoritmo que leva a solução para um melhor ou pior resultado. Cabe salientar que algumas vezes este tipo de abordagem pode não funcionar.

Métodos heurísticos têm uma crescente utilização nas ciências administrativas pois fornecem maneiras de tratar certos tipos de problemas que são computacionalmente mais complexos e que necessitam de um tempo de processamento muito grande, não justificando aplicações de otimizações, pois exigiriam recursos temporais e financeiros que superariam os ganhos obtidos. Exemplos destes problemas são a programação de sessões em eventos ou conferências muito grandes, a programação de serviços de transporte, entre outros (PIDD, 1998).

A idéia básica da abordagem heurística pode ser exemplificado pelas propostas de Simon e Newell (1958). A abordagem desta proposta parte do conceito da busca seqüencial, a partir de uma área de possíveis decisões. Simon (1978) afirma que, para chegar a uma solução, o tomador de decisão deve realizar várias

escolhas que irão constituir a solução espacial do problema. O autor ainda afirma que o tomador de decisão não tem todas as informações para tomar uma decisão ótima, então ele começa tomando decisões em cada etapa (este processo pode demorar), aplicando sempre uma heurística. A busca irá terminar quando um ponto suficientemente bom for encontrado.

A idéia de heurística, como proposto por Simon e Newell (1958), era que esta abordagem deveria possibilitar o uso do bom senso através do qual algumas escolhas poderiam ser realizadas à medida que o tomador de decisões evolui na rede de opções.

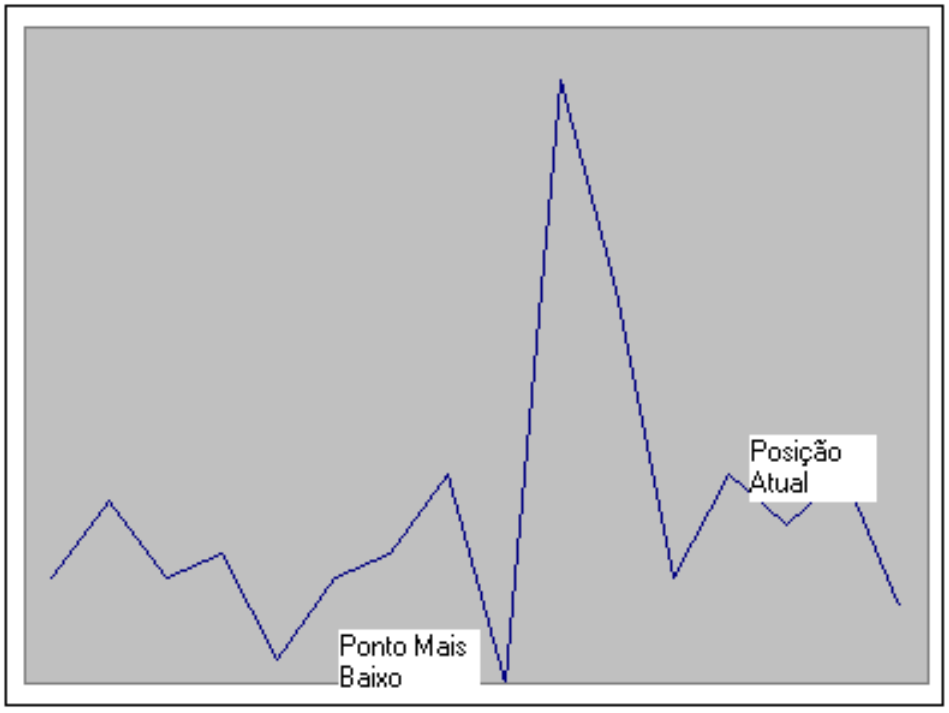
Simon (1978) estava especificamente interessado no processo de tomada de decisão do jogo de xadrez. Ele sugeriu que jogadores de xadrez experientes desenvolvem heurísticas que os permitem examinar rapidamente as opções possíveis quando confrontados com posições conhecidas no tabuleiro. Assim, ele argumentou, jogadores experientes (e vitoriosos) aprendem a reconhecer campos de situações similares e desenvolvem heurísticas para lidar com as mesmas.

Conforme Pidd (1998), outra analogia consiste de uma pesquisa exploratória, no escuro, buscando-se determinar o vale mais profundo em um terreno que não está representado em nenhum mapa. (este problema é equivalente a tentar encontrar uma solução com custo mínimo para determinado problema). Uma estratégia possível seria dirigir-se para baixo enquanto for possível – isto pode ser feito no escuro pelo contato com a terra (sentimento de declive). Seria bastante simples se a exploradora estivesse em um terreno contendo apenas um vale. Nestas circunstâncias, tudo o que ela precisa fazer é descer lentamente a ladeira, até que sinta que está no final do vale. Assim, ela pode concluir corretamente que se encontra no ponto mais profundo do vale.

Entretanto, na maioria das explorações irá existir mais de um vale e provavelmente alguns destes estarão acima ou abaixo de outro, ou seja, terão profundidades diferentes. Este problema está representado em duas dimensões na figura a seguir. Nesta figura, nossa exploradora encontrou o ponto mais profundo do terreno. O problema para a exploradora é encontrar alguma maneira de escalar as colinas, a fim de encontrar o caminho para a colina mais profunda. Nesta figura, temos uma grande vantagem sobre a exploradora: nós podemos ver onde ela precisa chegar, ou seja, temos o conhecimento do ponto objetivo. Porém, se estivéssemos em seu lugar, mesmo tendo o conhecimento, estaríamos nos

“debatendo no escuro”. Os métodos heurísticos procuram fornecer estratégias que possibilitem que uma busca seja realizada até que tenhamos uma certa garantia de que estamos no ponto mais profundo. Observe que não teremos certeza da solução, a menos que tenhamos conhecimento sobre a forma do terreno – e, se tivermos esta informação, provavelmente não será necessário realizar uma busca da maneira proposta acima. Se tivéssemos um problema de maximização, a analogia seria relativa a encontrar o ponto mais alto dos vales.

Figura 1: Explorando uma região no escuro



Fonte: Adaptado de Pidd (1996, p.284).

3.1 POR QUE UTILIZAR HEURÍSTICAS ?

Temos dois argumentos para responder a esta pergunta. A primeira é fazendo uma comparação entre métodos de otimização, como, por exemplo, a programação linear, onde o modelo do problema deve ser resolvido com equações lineares para a função objetivo e para as restrições. Por vezes, o modelo fica tão distante da realidade que perde o sentido da solução.

A segunda razão está relacionada a demanda computacional relativa a muitos problemas de otimização. Considere um problema de planejamento de um material laminado com 10 camadas internas distintas, encaixadas entre camadas de revestimento externo. Se existe a possibilidade de alocação de todas as camadas em todas as operações possíveis, ou seja, sua ordem é irrelevante, existem 3.628.800 (10 x 8 x 7 x ... x 2 x 1) permutações. Se o número de camadas aumentar para 12, o número de permutações aumenta para mais de 479 milhões. O problema, nesta questão, é que o número de opções aumenta exponencialmente à medida que o número de camadas cresce. Assim, para problemas muito grandes, mesmo que o modelo seja uma boa representação do mundo real, e mesmo que tenhamos um bom potencial computacional disponível, o tempo para solução do problema é tão grande que inviabiliza a utilização de métodos de otimização. Isto é particularmente verdade para situações nas quais mudanças freqüentes ocorrem com relação ao número e tipos de opções que estão sendo comparadas. Nestas situações, o único meio praticável para resolução do problema pode ser o uso da abordagem heurística.

Campelo e Maculan (1994) ratificam estas informações afirmando que, mesmo para problemas de porte pouco significativo, o número de passos, ou seja, a quantidade de instruções a serem processadas, será tão grande que o computador mais avançado levaria séculos no processo de cálculo. Exemplificando, em um problema cujo número de soluções viáveis é $n!$, admitindo que cada uma delas possa ser examinada em 10^{-9} segundos (1 nanosegundo), tem-se:

Quadro 1: Quadro comparativo entre número de tarefas e tempo de execução

n	Tempo de Execução
20	80 anos
21	1.680 anos

Fonte: Adaptado de Campelo; Maculan (1994)

Este trabalho se focará na meta heurística Busca Tabu para desenvolver um algoritmo de escalonamento. Esta estratégia vem recebendo a atenção de pesquisadores, com vários trabalhos já desenvolvidos cujos objetivos são semelhantes a este trabalho, vide: Aragão e Palmeira (1996); Valls, Perez e Quintanilla (1998); Wathier (1999); Müller (1993), etc... .

O algoritmo apresentado é uma adaptação do algoritmo proposto por Müller (1993).

4 BUSCA TABU

Busca Tabu é uma meta heurística proposta por Fred Glover em 1986 para resolução de problemas de otimização. Está baseada em princípios de solução inteligente de problemas. A Busca Tabu possui uma habilidade de guiar o processo de busca de métodos iterativos de melhoria. Neste contexto, a Busca Tabu fornece meios de explorar o espaço de soluções além dos pontos em que a heurística que ela está guiando, por exemplo, um procedimento de busca local encontraria ótimos locais. O processo no qual a Busca Tabu procura transcender a posição ótima local se baseia em uma função de avaliação que escolhe, a cada iteração, o movimento com maior valor de avaliação.

Para descrever como a Busca Tabu trabalha, apresentamos um problema de otimização combinatória no seguinte formato:

$$\text{(P)} \quad \text{Minimize } c(x): x \in X \text{ in } R_n$$

A função objetivo $c(x)$ pode ser linear ou não linear, e a condição $x \in X$ é assumida por restringir elementos específicos de x para valores discretos. Em adição, configurações (P) precisam representar um formato alterado do problema inicial, onde X é um conjunto de vetores que normalmente são qualificados como factíveis, e $c(x)$ é uma função de penalização, designada para garantir que a solução ótima para (P) seja igualmente ótima para problema do qual originou.

Conforme Glover (1990), os passos de um processo de Busca Tabu genéricos são mostrados abaixo:

a) passo 1: escolhe a solução inicial;
b) passo 2: seleciona a melhor solução admissível, S_{best} (onde: S_{best} é a melhor solução entre todas $S' \notin N(S) : S'$ não pertença a lista tabu);
c) passo 3: atualiza a solução corrente $S \leftarrow S_{best}$, e também atualiza a lista tabu;
d) passo 4: se o critério de parada foi satisfeito, pare, caso contrário vá para o passo 2.

O cerne deste processo se encontra nos passos 2 e 3. O melhor movimento admissível é aquele com maior avaliação na vizinhança da solução corrente em termos de valor da função objetivo e restrições tabu. A função de avaliação escolhe o movimento que produz a maior melhoria, ou a menor piora na função objetivo. A lista tabu é introduzida no sentido de guardar características dos movimentos realizados, para que futuros movimentos que apresentam estas mesmas características possam ser classificados de tabu (isto é, proibidos de serem executados). A aceitação de movimentos que piorem o resultado final abre a possibilidade de retorno a soluções já visitadas, portanto, ciclos podem ocorrer e a função da lista tabu é evitar que tais ciclos ocorram. Desse modo é necessário restringir a busca através de uma estratégia de proibição e evitar que seqüências de soluções sejam repetidas e, com isto, induzir a exploração de novas regiões.

Busca Tabu é uma meta-heurística, conforme Glover (1989) composta pelas estratégias que se seguem.

4.1 ESTRATÉGIA DE PROIBIÇÃO

Gerencia o que será incluído na lista tabu. O objetivo desta estratégia é estabelecer um mecanismo que proíba certos movimentos (faça-os receber o status tabu), no sentido de evitar o retorno à mesma situação. A lista tabu armazena atributos dos movimentos que foram realizados. Supõe-se que a probabilidade do retorno à mesma situação é inversamente proporcional a distância entre a solução corrente e as soluções anteriores. Assim, para evitar a escolha de movimentos que representam o reverso de qualquer decisão tomada durante as últimas $|T_s|$ iterações anteriores. Normalmente $|T_s|$ é chamado de tamanho da lista tabu.

4.2 ESTRATÉGIA DE LIBERAÇÃO

Gerencia o que será retirado da lista tabu e em que momento isto ocorrerá. A Busca Tabu remove as restrições, tabu, de um movimento, de modo que ele poderá ser escolhido a partir daquele momento. Os atributos de um movimento tabu permanecem na lista durante $|T_s|$ iterações. Se o tempo de permanência dos atributos na lista se esgota, então eles são liberados do seu status tabu por sua estratégia.

4.3 CRITÉRIO DE ASPIRAÇÃO

É uma maneira de retirar o status tabu de um movimento caso ele se apresente com características muito boas e não ocasione retorno a uma mesma situação. Um movimento é admissível se suas restrições tabu não forem violadas, entretanto, ele também pode ser admissível se o critério de aspiração retirar o seu status tabu. Observe que, sendo os atributos aproximações, um movimento pode ser proibido apenas se nunca tiver sido realizado anteriormente. Em certas situações fica evidente que o movimento, embora esteja na Lista Tabu, é benéfico, (por exemplo, se a solução candidata tem um valor de função objetivo melhor que a melhor solução corrente) e o critério de aspiração procura sanar estas distorções.

4.4 MEMÓRIA DE CURTO PRAZO

É a base de um algoritmo de Busca Tabu. Esta estratégia de memória de curto prazo busca uma integração entre as estratégias de proibição, liberação, critério de aspiração e estratégia de seleção do movimento.

4.5 MEMÓRIAS DE MÉDIO E LONGO PRAZO

São estratégias de aprendizado que podem ser utilizadas durante a execução da Busca Tabu no intuito de realizar uma busca mais minuciosa sobre uma região supostamente promissora (intensificação) ou mesmo guiar a busca para regiões inexploradas do espaço de soluções (diversificação).

4.6 CRITÉRIO DE PARADA

Normalmente a Busca Tabu para após um número pré-especificado de iterações ou após um número pré-especificado de iterações após a última melhoria ter sido encontrada.

Wathier (1999) apresentou um algoritmo, adaptado de Müller (1993), utilizando a meta-heurística Busca Tabu para o escalonamento de tarefas em processadores paralelos aplicável em problemas de processamento distribuída.

Este algoritmo está dividido em 3 partes, a saber: Alocação Inicial, Fase Tabu e Pós-Otimização. Como o problema apresentado pelo autor é semelhante ao apresentado neste trabalho, ou seja, ambos tratam da alocação de tarefas em processadores paralelos, onde, no trabalho apresentado pelo autor, uma parte da tarefa começa e termina no mesmo processador, fez-se uma abstração deste conceito para o entendimento de que uma tarefa começa e termina por completo no mesmo processador, sendo assim possível aplicar este algoritmo para o objeto de estudo deste trabalho.

5 OBJETIVOS

De modo a responder a questão proposta na pesquisa, foram estabelecidos o objetivo geral e os objetivos específicos, abaixo mostrados.

5.1 OBJETIVO GERAL

Aplicar um algoritmo baseado na meta-heurística Busca Tabu para o problema de alocação de tarefas em processadores paralelos, onde cada tarefa tem por caraterística iniciar e terminar no mesmo processador, sem interrupções, com tempo de *setup* entre tarefas dependente da seqüência das tarefas.

5.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos do trabalho são:

a) desenvolver um algoritmo baseado na busca tabu para reduzir o *makespan* do problema de sequenciamento de tarefas em processadores paralelos com tempo de *setup* dependente da seqüência de execução das tarefas, sem comprometer as datas de entregas (*due-date*);
b) fazer uma análise dos resultados obtidos no trabalho com dados reais já levantados em uma empresa do setor calçadista;
c) propor implementações no algoritmo final, baseadas nas necessidades reais dos escalonadores humanos, para a efetiva utilização e aplicação do algoritmo, no formato de um programa de computador, em ambientes de produção.

6 METODOLOGIA

Para Flood e Carson (1990), há uma diferença entre filosofia, metodologia e técnica. A filosofia seria a diretriz para a ação, a metodologia seria uma direção para a ação e a técnica um programa específico de ação. A metodologia apresenta uma constituição para ações que serão executadas e que podem transformar-se numa estratégia possível de ser realizada. Isso permite ao pesquisador personalizar e adaptar a estratégia de acordo com as mudanças percebidas.

Para o desenvolvimento dos algoritmos propostos neste trabalho, avaliou-se o problema de sequenciamento da indústria em questão, onde há *setup* entre modelos de construções diferentes.

Este trabalho usou como base a metodologia de resolução de problemas desenvolvida no âmbito da pesquisa operacional, apresentada por Ackoff (1977), com a proposta de usar um modelo matemático para avaliar a eficiência dos algoritmos propostos e sua efetiva aplicação. Foram utilizadas as etapas clássicas de um estudo de pesquisa operacional, como caminho referencial na condução do estudo, de acordo com Taha (1982) e Wagner (1986).

6.1 FORMULAÇÃO DO PROBLEMA

O problema existente é a necessidade da redução do *makespan* no sequenciamento de tarefas em processadores paralelos, com tempo de *setup* dependente das tarefas. O objetivo é poder atender mais tarefas em menos tempo, para poder aceitar mais tarefas.

6.2 CONSTRUÇÃO DO MODELO

Neste estudo, a construção do modelo foi baseada no algoritmo proposto por Müller (1993) para a alocação de tarefas em processadores paralelos independentes onde cada tarefa começa e termina no mesmo processador sem interrupção. Este algoritmo foi escolhido por apresentar bons resultados em outros trabalhos, vide Wathier (1999). Após a definição do algoritmo utilizado a meta heurística Busca Tabu, foi construído um modelo matemático utilizando a Programação Linear Inteira, para avaliar o desempenho do algoritmo. Foram feitos vários testes com diversas situações de número de tarefas e número de processadores.

6.3 SOLUÇÃO DO MODELO

O algoritmo com a meta-heurística Busca Tabu foi desenvolvido utilizando a linguagem de programação *CLIPPER* fornecida pela *Computer Associates* para *MS-DOS*.

6.4 VALIDAÇÃO DO MODELO

Foi desenvolvido um modelo matemático de Programação Linear Inteira para resolver o problema e comparar com o algoritmo utilizando a meta-heurística Busca Tabu. Este modelo matemático de Programação Linear Inteira foi resolvido utilizando o *software LINDO* da *LINDO SYSTEMS*.

6.5 AVALIAÇÃO FINAL

Após comparar os resultados do algoritmo e o do modelo de Programação Linear Inteira, os resultados obtidos nos levam para a aplicação do algoritmo em problemas maiores e que se aplicam no contexto em questão.

7 MODELAGEM

O problema ao qual este trabalho está focado é a alocação de n tarefas em m processadores, com o objetivo de reduzir o *makespan* sem aumentar o atraso geral de entrega das tarefas. Na indústria em questão, trabalha-se com um estoque de pedidos de produtos, ou seja, se produz somente para entrega e não para estoque, com pedidos feitos antecipadamente pelos clientes. O atendimento de novas solicitações de pedidos e produtos pelos clientes depende da disponibilidade de recursos nas linhas de produção (processadores) da empresa. Se for possível reduzir o *makespan*, mais pedidos poderão ser atendidos, gerando um maior faturamento para a empresa.

Neste projeto, será trabalhado com n tarefas (*jobs*), onde cada tarefa contém um d número de dias de carga no processador. Para o nosso exercício, assume-se que os processadores são idênticos, ou seja, tem a mesma capacidade instalada. Com isto cada tarefa ocupa o mesmo tempo de carga em cada processador.

Assume-se que a capacidade instalada e a definição da capacidade de cada processador estão definidas de forma correta pela empresa que fornece os dados para o trabalho. Estas informações são de conhecimento do modelador e, para este trabalho, são fixas.

Observação: Na indústria em questão, os processadores (linhas de produção) não têm uma capacidade instalada fixa por processador. A capacidade instalada de cada linha (processador) varia de acordo com a necessidade de entrega da tarefa (*job*), e com a quantidade de recursos disponíveis. Na prática, a empresa como um todo tem um estoque de recursos, máquinas e equipamentos a disposição, que são alocados em linhas de produção conforme a necessidade. Embora esta colocação, o presente trabalho assume que cada linha de produção tenha uma capacidade instalada fixa.

Cada tarefa tem as seguintes características:

a) número da tarefa, T ;
b) número de dias necessários na linha de produção (no processador), d ;
c) data limite de entrega, L ;
d) modelo do produto a ser produzido;

e) construção do modelo do produto a ser produzido. produtos de mesma construção não tem tempo de *setup* (preparação de máquinas e ferramentas) no final de uma tarefa e início de outra. a tarefa é acrescida de um dia para o tempo de *setup* quando encerra uma tarefa de uma construção e inicia outra tarefa de outra construção;
f) custo de R$ 10,00 (dez reais) por dia de atraso da tarefa.

7.1 A SOLUÇÃO PROPOSTA USANDO BUSCA TABU

A solução proposta é uma adaptação do modelo apresentado por Müller (1993), que é dividido em 3 etapas, conforme serão expostas

a) etapa 1 - Alocação Inicial;
b) etapa 2 - Fase Tabu;
c) etapa 3 - Fase Pós-Otimização.

As fases deste algoritmo serão detalhadas a seguir.

7.1.1 Algoritmo Proposto – Redução do Makespan

Apresentamos um algoritmo para a solução inicial de alocação de tarefas nos processadores. A partir desta solução, será aplicada a Busca Tabu para tentar reduzir o *makespan*.

7.1.1.1 Alocação Inicial

Na Alocação Inicial (Etapa 1), as n tarefas são inseridas nos P processadores de uma maneira gulosa (inserção no processador que cause o menor acréscimo no tempo de finalização);

a) inicializa todos os processadores com 0 (zero) tarefas, $t = 0$.
b) ordena as tarefas por data de entrega, em ordem crescente. para tarefas com a mesma data de entrega, o segundo critério de ordenação é o tempo necessário no processador, também crescente.
c) para cada tarefa, localize o processador com menor carga dentre os processadores que podem realizar a tarefa que tem o menor acréscimo de setup.

Guarde esta solução inicial para a Fase Tabu.

7.1.1.2 Fase Tabu

Após a alocação inicial, na fase tabu, identifica-se o processador mais carregado. então, para cada tarefa deste processador, o algoritmo verifica as mudanças causadas no tempo máximo de finalização pela retirada e reinserção em outro processador. depois disto, ela escolhe o movimento que cause o mínimo tempo máximo de finalização e não seja considerado tabu e que não aumente o atraso geral. este procedimento é repetido por um número pré-determinado de iterações sem melhoria na solução incumbente.

Suponha que M é o processador mais carregado, ou seja, com maior tempo de finalização.

$C_{M_i J_j}$ = tempo de finalização (tempo necessário para finalizar todas as tarefas) do processador M_i se a tarefa J_j é inserida nele.

$C_{\overline{M_i}J_j}$ = tempo de finalização do processador $\overline{M_i}$ se a tarefa J_j é retirada dele.

Passo 1: Faça o contador de iterações $t = 0$

Passo 2: Para cada tarefa J_j do processador mais carregado, encontre:

a) o conjunto de processadores candidatos a receber a tarefa selecionada, sendo que o processador candidato tem que possuir, pelo menos, uma tarefa predecessora ou sucessora global da tarefa selecionada, ou seja, uma tarefa da mesma construção da tarefa selecionada, com o objetivo de alocar neste processador, e reduzir o tempo de *setup*, sendo: $M(J_j) = \{M_i \mid M_i \neq \overline{M_i}\}$ *e* M_i incluir entre as suas tarefas pelo menos uma sucessora global ou uma predecessora global de J_j.

b) o menor tempo de inserção da tarefa selecionada J_j em um dos processadores selecionados, $M(J_j)$, definido como:

$$C_{\overline{M}_i J_j} = \min_{M_i \in M(J_i)} \{C_{M_i J_j}\}$$

c) o tempo máximo de finalização, se a tarefa selecionada for transferida do processador mais carregado $(\overline{M}_j)$ para o processador candidato (M_i^*), definido como:

$$T_{J_j} = MAX\{C_{\overline{M}_i J_j}, C_{M_i^* J_j}\}$$

Passo 3: Escolha a tarefa a ser retirada do processador mais carregado como sendo a tarefa que gera o menor acréscimo e que não seja tabu (a menos que atenda o critério de aspiração).

Então, retire J_j^* de $\overline{M_i}$ e insira em M_i^*

Proíba o movimento reverso por θ iterações.

Passo 4: Identifique o processador mais carregado.

Passo 5: Incremente o contador de iterações em 1 (um), $t = t + 1$.

Se o número máximo de iterações sem melhoria da solução não for alcançado, vá para o PASSO 2. Caso contrário vá para a FASE DE PÓS-OTIMIZAÇÃO.

7.1.1.3 Fase Pós-Otimização

Na fase pós-otimização, devido às várias inserções e retiradas feitas nos processadores, ao final da fase tabu é possível ocorrerem casos em que retirando e reinserindo uma tarefa no mesmo processador haja uma diminuição no seu tempo de finalização (o que pode levar a uma diminuição no tempo máximo de finalização). isto é causado devido às características de inserção e retirada, que realizam uma otimização local a cada operação.

Passo 1: Identifique o processador mais carregado, sendo $\overline{M}_i$ e o tempo de finalização deste processador, sendo C_{M_i}

Passo 2: Para cada uma das tarefas alocadas neste processador, realize uma retirada e uma reinserção no mesmo processador gerando um novo tempo de finalização, sendo $\overline{C}_{M_i}$.

Passo 3: Se o tempo mínimo de finalização não se altera então PARE. Se o tempo mínimo de finalização reduz, atualize e vá para o PASSO 1.

O próximo capítulo descreve o processo de validação do modelo.

8 VALIDAÇÃO

Como ocorre em qualquer processo de modelagem, e esta implementação não poderia ser diferente, há a necessidade do sistema ser validado para ter valor de uso. Validar um modelo significa construir o modelo certo. No processo de validação de um modelo é importante averiguar o comportamento da implementação computacional em relação a realidade e a modelagem. Neste sentido, a validação pode ser considerada um passo com caráter mais cientifico e efetivo para o desenvolvimento de sistemas computacionais. Sem validação formal, os modelos podem causar erros caros.

Validação é o processo de definir se o comportamento de um determinado modelo representa o mundo real no domínio de um problema particular.

A validação conceitual assegura que as suposições e teorias em que está baseado o modelo conceitual são pertinentes ao tipo de problema. O modelo conceitual pode ser definido como o modo pelo qual modeladores e usuários percebem e apresentam o problema. Conforme Miser e Quade (1988), a verificação é definida como sendo um processo que tem a finalidade de provar até que ponto um modelo é fidedigno a sua concepção, e se isso os torna válidos. A concretização é definida como "[...] a demonstração que um modelo computacional, dentro de seu domínio de aplicabilidade, possui um alcance satisfatório de precisão consistente com a aplicação intencional do modelo (BALCI, 1981, p.190).”.

Definida nossa aproximação com a validação, é possível estabelecer-se as linhas genéricas da validação, segundo Gass (1983), é importe conhecer as forças e fraquezas do sistema; conhecer os limites das capacidades de previsão e de aconselhamento do sistema; e instruir e justificar as suposições do sistema às expectativas dos usuários. Este processo é uma tarefa trabalhosa que demanda muito tempo, muito esforço na busca do conhecimento e custos financeiros, e ainda não garante um modelo computacional livre de erros, pois é impossível enumerar previamente os cenários e informações de todos os usuários, afirma Jafar e Bahill (1993).

Para a validação foram utilizados dados reais de uma indústria do ramo calçadista. Foram feitas comparações entre os resultados obtidos com a Programação Linear Inteira e o algoritmo utilizando a busca tabu.

8.1 O MODELO DE PROGRAMAÇÃO LINEAR INTEIRA

O modelo de programação linear inteira compreende os seguintes tópicos: notação, variáveis de decisão, função, objetivo e restrições.

8.1.1 NOTAÇÃO

- considere P como o conjunto de processadores;
- considere T como o conjunto de tarefas;
- considere O como o conjunto de ordens de execução possíveis para cada tarefa;
- considere S como o conjunto de tarefas predecessoras possíveis para cada tarefa, em cada ordem de execução;
- considere $p = 1,2,...,P$;
- considere $t = 1,2,...,T$;
- considere $o = 1,2,...,O$;
- considere $s = 1,2,...,S$;
- considere d_t como o tempo de execução da tarefa t onde $t = 1,2,...,T$, já adicionado o tempo de *setup*, quando a construção da operação predecessora do mesmo processador for diferente da construção da tarefa t.
- considere d como um coeficiente que determina o tempo da tarefa t no processador p, quando executada na ordem o, sendo precedida pela tarefa s se $o > 1$

8.1.2 VARIÁVEIS DE DECISÃO

- considere TT_p como o tempo total de processamento de cada processador p, para $p = 1,2,...,P$;
- considere YT_p como $TT_p - TT_{p+1}$ quando $TT_p <= TT_{p+1}$ para $p = 1,2,...P-1$.
- considere NT_p como $TT_p - TT_{p+1}$ quando $TT_p >= TT_{p+1}$ para $p = 1,2,...P-1$.
- considere TP_{tpos} com valor 1 quando a execução da tarefa t estiver alocada no processador p na ordem de operação o, precedida pela tarefa s quando $o > 1$; caso contrário 0 (zero), como:

$$TP_{tpos} = \begin{cases} 1, t \text{ estiver alocada no processador } p \text{ na ordem de operação } o, \text{ precedida pela tarefa } s, \text{ quando } o>1 \\ 0, \text{ caso contrário.} \end{cases}$$

8.1.3 FUNÇÃO OBJETIVO

$$Min \sum_{p=1}^{P} TT_p + YT_p + NT_p$$

8.1.4 RESTRIÇÕES

a) atribui valor à variável TTp, com o tempo total de processamento de cada processador, sendo:

$$\sum_{t=1}^{T} \sum_{o=1}^{O} \sum_{s=1, s \neq t}^{S} d\left(TP_{tpos}\right) - TT_p = 0, \text{ para } p = 1,2,...P$$

b) atribui valor à variável YT_p, com a diferença entre dois processadores, quando o tempo total de processamento do primeiro processador for menor ou igual ao tempo total de processamento do segundo processador, sendo:

$$TT_p - TT_{p+1} + YT_p <= 0, \text{ para } p = 1,2,...P-1$$

c) atribui valor à variável NT_p, com a diferença entre dois processadores, quando o tempo total de processamento do primeiro processador for maior ou igual ao tempo total de processamento do segundo processador, sendo:

$$TT_p - TT_{p+1} - NT_p >= 0, \text{ para } p = 1,2,...P-1$$

d) cada tarefa t somente pode ser executada uma vez entre todos os processadores, sendo:

$$\sum_{p=1}^{P} \sum_{o=1}^{O} \sum_{s=1, s \neq t}^{S} TP_{tpos} = 1, \text{ para } t = 1,2,...,T$$

e) as seqüências de operações devem estar preenchidas de tal forma, que o sistema sugira a primeira operação, a segunda operação, a terceira operação, e assim por diante, para cada processador, sem falhar nenhuma ordem de operação, sendo:

$$\sum_{p=1}^{P} \sum_{t=1}^{T} \sum_{s=1, s \neq t}^{S} TP_{tpos} - \sum_{t'=1, t' \neq t}^{T} \sum_{s=1, s \neq t'}^{S} TP_{t'p(o+1)s} >= 0, \text{ para } o = 1,2,...,O-1$$

f) a tarefa, quando for executada na segunda operação do processador em diante, deve ser precedida pela tarefa que indica no *setup* da notação, sendo:

$$\sum_{p=1}^{P} \sum_{o=1}^{O-1} \sum_{s=1, s \neq t}^{S} TP_{tpos} - \sum_{t'=1, t' \neq t}^{T} TP_{t'p(o+1)t} >= 0, \text{ para } t = 1,2,...T$$

Todas as variáveis são binárias.

A tabela a seguir apresenta uma comparação entre o modelo de programação linear inteira (PLI), e o algoritmo utilizando a estratégia Busca Tabu adaptado do modelo apresentado por Müller (1993) e descrito anteriormente:

Quadro 2 :Comparação da PLI e Método Busca Tabu

Tarefas	Processadores	Programação Linear				Busca Tabu		Resultado
		Variáveis Inteiras	*Make span*	Iterações	Branches	*Make span*	Iterações	
3	2	40	3	39	2	3	0	Ótimo
4	2	100	4	79	2	4	1	Ótimo
4	3	153	3	17.091	500	3	0	Ótimo
5	2	204	5	15.556	414	5	0	Ótimo
5	3	309	3	3.911	76	3	0	Ótimo
5	4	416	3	1.090.806	10.910	3	1	Ótimo
6	2	364	6	305.924	2.730	6	1	Ótimo
6	3	549	4	7.331.273	43.950	4	1	Ótimo
6	4	736	Não Terminou					
6	5	925	Não Terminou					
7	2	592	7	66.448	391	7	0	Ótimo
7	3	891	Não Terminou					
7	4	1.192	Não Terminou					
7	5	1.800	Não Terminou					
7	6	1.353	Não Terminou					
8	2	900	8	794.721	1.770	8	1	Ótimo
8	3	1.353	Não Terminou					
8	4	1.808	Não Terminou					
8	5	2.265	4	5.289.949	15.197	4	0	Ótimo
8	6	2.724	N/A	6.666.851	22.789	4	0	Bom
8	7	3.185	Não Terminou					
10	2	1.804	9	661.007	1.993	10	0	Bom

Fonte: Elaborado pelo autor.

No quadro apresentado acima, 83% das ocorrências que terminaram utilizando a Programação Linear Inteira tiveram um resultado ótimo no algoritmo utilizando a Busca Tabu. Os testes foram efetuados em um computador *Pentium*, 350 Mhz. Para a solução da Programação Linear Inteira foi utilizando o *software*

Lindo. O tempo de processamento do *software Lindo* para problemas com mais de 66.000 (sessenta mil iterações) levam 15 minutos, chegando a mais de 2 dias com problemas com mais de 3.000.000 (três milhões de iterações).

Observa-se o pequeno número de iterações do algoritmo utilizando a Busca Tabu. Na maioria dos casos, a fase de alocação inicial se mostra suficiente para a alocação das tarefas, não necessitando ajustes na fase tabu ou na pós-otimização.

Todos os problemas processados com o algoritmo utilizando a Busca Tabu levaram no máximo 5 (cinco) minutos.

9 AVALIAÇÕES

Para a avaliação do algoritmo foram utilizadas informações colhidas junto a indústria para simular situações mais próximas da realidade. Desde a configuração das unidades fabris e das linhas de produção, até as informações das tarefas, prazos de entrega, tempo das tarefas nos processadores e o escalonamento feito manualmente foram obtidas junta a indústria em questão para a aplicação do algoritmo, e a sua avaliação com os resultados obtidos manualmente.

9.1 DESCRIÇÃO DA INDÚSTRIA

A indústria está dividida em 6 unidades, sendo 3 unidades na mesma cidade, e as demais em 3 cidades diferentes. Toda a distribuição dos pedidos para produção é processada na mesma unidade, ou seja, o modelador é único e centraliza o processo.

O diagrama das unidades fabris e das linhas de produção é apresentado no quadro abaixo:

Quadro 3: Diagrama das linhas de produção

Unidade 1	Linha de produção 1
	Linha de produção 2
Unidade 1	Linha de produção 3
	Linha de produção 4
	Linha de produção 5
Unidade 2	Linha de produção 6
	Linha de produção 7
Unidade 3	Linha de produção 8
Unidade 4	Linha de produção 9

	Linha de produção 10
Unidade 5	Linha de produção 11
Unidade 6	Linha de produção 12

Fonte: Fornecido pela Indústria

Como já foi dito anteriormente, a fábrica possui um estoque de recursos à disposição para regular a capacidade instalada de cada linha de produção. Portanto, é comum o número de linhas de produção em cada unidade se alterarem para mais ou para menos, com muita rapidez, sempre com o objetivo de atender as solicitações do mercado.

9.2 LEVANTAMENTO DOS DADOS

Os dados utilizados para os testes deste trabalho foram recolhidos junto a empresa onde está sendo aplicado este algoritmo. Informações históricas foram repassadas com o objetivo de comparar os resultados obtidos pelo modelador com os resultados gerados pelo algoritmo apresentado.

O quadro abaixo apresenta situações reais dos resultados que o modelador obteve e utilizou na fábrica para a produção.

Quadro 4: Comparação do processador mais carregado entre o *MakeSpan* da fábrica e o *MakeSpan* do algoritmo proposto.

Tarefas	Proc.	*Makspan* da Fábrica (Proc.Mais Carregado)	Tempo de Processamento do Algoritmo	*Makespan* do algoritmo	% Redução do *Makespan* (Proc. Mais Carregado
450	12	107	00:04:20	84	21%
446	12	106	00:03:05	81	24%
446	12	106	00:02:03	81	24%
425	12	106	00:02:05	78	26%
425	12	100	00:03:38	75	25%
431	13	100	00:01:32	73	27%
425	12	106	00:03:34	75	29%
437	13	106	00:01:01	70	34%
444	13	106	00:03:56	71	33%

446	13	101	00:02:36	73	28%
448	13	100	00:00:43	70	30%
451	13	100	00:01:53	72	28%
442	13	100	00:02:08	77	23%
407	13	100	00:01:18	69	31%
409	13	100	00:01:24	77	23%
411	13	100	00:01:17	68	32%
418	13	100	00:01:30	67	33%
422	13	100	00:03:26	70	30%
444	13	100	00:02:39	81	19%
429	13	100	00:02:12	72	28%
436	13	100	00:01:47	73	27%
360	13	98	00:01:09	57	42%
450	12	98	00:02:25	84	14%
446	12	98	00:01:01	81	17%
411	13	95	00:01:38	68	28%
418	13	95	00:02:01	67	30%
422	13	95	00:05:18	70	26%
420	13	93	00:02:02	67	28%
431	13	93	00:02:07	75	19%
427	13	93	00:01:06	76	18%
444	13	93	00:03:07	81	13%
429	13	106	00:04:29	72	32%
436	13	105	00:02:44	73	30%

Fonte: Elaborado pelo Autor

Avaliando-se o quadro acima, percebe-se que mudanças são feitas na programação geral da fábrica, mas o *makespan* do processador mais carregado não se altera. Devido a este fato, observou-se que, em muitos casos, o processador mais carregado, tem muito mais carga que os outros processadores (vide anexo 3). Então fez-se nova avaliação da carga dos processadores, considerando-se o *makespan* médio, para novamente avaliar o desempenho do algoritmo. Os resultados estão no quadro abaixo.

Quadro 5: Comparação da carga média dos processadores entre o *MakeSpan* da fábrica e o *MakeSpan* do algoritmo proposto.

Tarefas	Proc.	Makespan (Média dos Proc.)	Tempo de Processamento do Algoritmo	*Makespan* Médio do algoritmo	% Redução do *Makespan* (Média dos Proc.)
450	12	82	00:04:20	79	4%
446	12	81	00:03:05	79	2%

446	12	81	00:02:03	75	7%
425	12	81	00:02:05	73	10%
425	12	79	00:03:38	75	5%
431	13	79	00:01:32	73	8%
425	12	79	00:03:34	68	14%
437	13	79	00:01:01	68	14%
444	13	73	00:03:56	68	7%
446	13	72	00:02:36	69	4%
448	13	72	00:00:43	70	3%
451	13	72	00:01:53	71	1%
442	13	72	00:02:08	67	7%
407	13	73	00:01:18	67	8%
409	13	73	00:01:24	67	8%
411	13	74	00:01:17	66	11%
418	13	74	00:01:30	67	9%
422	13	74	00:03:26	75	-1%
444	13	74	00:02:39	70	5%
429	13	74	00:02:12	70	5%
436	13	75	00:01:47	55	27%
360	13	70	00:01:09	80	-14%
450	12	71	00:02:25	79	-11%
446	12	71	00:01:01	75	-6%
411	13	70	00:01:38	67	4%
418	13	70	00:02:01	66	6%
422	13	69	00:05:18	67	3%
420	13	69	00:02:02	66	4%
431	13	69	00:02:07	67	3%
427	13	79	00:01:06	71	10%
444	13	75	00:03:07	70	7%
429	13	75	00:04:29	71	5%
436	13	74	00:02:44	70	5%

Fonte: Elaborado pelo Autor

Atualmente, o processo de ajustes na programação da produção ocorre várias vezes durante a semana, com o objetivo de fazer melhorias nos prazos de entrega, encaixar novos pedidos ou efetuar correções motivadas por atrasos nas linhas de produção em relação ao que está programado, ou por algum outro motivo que gere a necessidade de reprogramação. Este trabalho é manual, e estafante, consumindo quase um dia inteiro de trabalho, sendo necessário o envolvimento de mais de uma pessoa.

Avaliando o *makespan* nos quadros acima, percebe-se que o processador mais carregado tem uma carga bastante superior a média das cargas de todos os

processadores. No geral o processador mais carregado tem uma carga 35% maior que a média das cargas de todos os processadores.

Conforme os resultados apurados no primeiro quadro, podemos perceber que temos uma significativa redução do *makespan* no processador mais carregado, com um tempo de processamento bastante reduzido, se comparado ao tempo dedicado para esta tarefa pelo modelador.

Em média, o *makespan* do processador mais carregado foi reduzido em 27% quando o algoritmo é aplicado para a resolução do problema. Na comparação da média de carga dos processadores, no segundo quadro, a redução foi de 5,3%.

Percebe-se, avaliando-se primeiro quadro, que em alguns casos, embora o número de tarefas se altere, o *makespan* gerado pelo escalonador humano não se altera. Provavelmente novas tarefas são alocadas, ou ajustes são efetuados nos processadores menos carregados. Conforme exemplificou Simon (1978), esta é uma heurística desenvolvida e aplicada com a experiência do profissional. O algoritmo utiliza este mesmo conceito na primeira etapa, ou seja, na fase de alocação inicial, alocando novas tarefas sempre no processador que gera menor acréscimo no *makespan*.

10 COMENTÁRIOS FINAIS

Tendo este trabalho o objetivo de aplicar um modelo para a redução do *makespan* no sequenciamento de tarefas em processadores paralelos, cabe ressaltar que o modelo apresentado deve ser considerado como um passo inicial para a efetiva aplicação em ambiente de produção.

Para a solução do problema, inicialmente foi apresentado um algoritmo, desenvolvido por Müller (1993) para o problema computacional de processamento distribuído. Este algoritmo foi adaptado para o problema deste trabalho. Dividido em três etapas, o algoritmo faz uma alocação inicial das tarefas nos processadores, um processamento chamado de Fase Tabu, onde implementa a meta-heurísitca Busca Tabu, e uma fase chamada de pós-otimização. Na alocação inicial, o algoritmo proposto tem as tarefas ordenadas em ordem crescente de data de atendimento ou data de entrega e nas Fases Tabu e Pós-Otimização, o algoritmo, nos processadores mais carregados, busca alocar em seqüência tarefas que tenham menor tempo de *setup* entre si, para reduzir o tempo de *setup* total, e consequentemente o *makespan*, avaliando se o movimento vai gerar aumento no atraso geral. Se for o caso de aumento do atraso geral, então o movimento não é efetuado.

O algoritmo foi validado fazendo-se comparações com a Programação Linear Inteira, com vários testes com situações diferentes entre si. Estas situações consideraram várias configurações diferentes entre número de tarefas e número de processadores, com o objetivo de apresentar o desempenho do algoritmo. Em comparação com a Programação Linear Inteira, o algoritmo obteve um resultado ótimo em 83% dos casos, sendo que nos outros casos, o resultado chegou bastante perto do ótimo.

O algoritmo foi aplicado utilizando casos reais recolhidos junto a indústria calçadista já citada anteriormente, para a avaliação e comparação dos resultados obtidos pelo escalonador manual e os resultados do algoritmo.

Embora o algoritmo proposto tenha reduzido o *makespan* dos processadores em um percentual considerável e que geraria muitos ganhos, é importante salientar

que a experiência, a capacidade de negociação e a percepção do ambiente por parte do escalonador humano não deve ser desprezada.

10.1 LIMITAÇÕES DO MODELO

O modelo proposto não leva em consideração alguns fatores, que atualmente entram na consideração do escalonamento humano. O algoritmo proposto sempre parte de um ponto onde todos os processadores estão com carga 0 (zero), ou seja, nenhuma tarefa alocada, então faz a alocação inicial. No modelo manual, os processadores já estão com todas as tarefas conhecidas alocadas, e alterações são efetuadas com as alocações de tarefas já definidas, ou seja, não é feito outro escalonamento inicial. As alterações no escalonamento ocorrem para ajustes da programação geral da fábrica, devido a diversos motivos, sendo alguns citados a seguir:

a) entrada de novos pedidos;
b) pelo atraso na produção;
c) atraso na entrega de matéria-prima;
d) quebra de máquina.

Neste momento, quando a alocação de tarefas está sendo revista, o escalonador humano leva em consideração algumas informações não utilizadas no modelo proposto, para citar algumas:

a) disponibilidade de matérias-primas;
b) o custo ou inconveniência de tirar uma tarefa alocada em unidade, e alocá-la em outra, gerando transporte de recursos e de matérias-primas já disponibilizadas;
c) a maior capacitação da mão-de-obra de cada unidade para execução de determinados serviços;

d) negociação do prazo de entrega com os clientes de alguns pedidos que serão atrasados, para tornar possível a nova situação do escalonamento.

10.2 RECOMENDAÇÕES

Este algoritmo, recebendo algumas novas implementações, com o objetivo de atender as limitações citadas anteriormente, poderia servir como uma ferramenta de auxílio para o processo de escalonamento, sendo de grande valia na execução do trabalho.

Para o melhor desempenho e uso do algoritmo, pode-se citar algumas melhorias ou implementações a serem feitas:

a) utilizar como alocação inicial o escalonamento já em produção, ao invés de a cada vez ser feito uma alocação inicial diferente;
b) avaliação do custo da troca da alocação das tarefas de uma unidade fabril para outra;
c) ser possível receber ajustes nas datas das tarefas por ocasião de atraso de produção;
d) avaliar disponibilidade de matérias-primas;
e) avaliar a maior capacidade de determinadas unidades fabris para determinados produtos.

Talvez resistências ao uso de uma ferramenta como esta surjam e devam ser superadas, já que seria necessário conseguir a confiança por parte dos usuários. A confiança é imprescindível para o efetivo uso e auxílio a uma tarefa tão nobre e importante dentro das organizações, que é a alocação de tarefas e consequentemente de recursos físicos, equipamentos, fluxo de caixa, recursos humanos, etc... .

REFERÊNCIAS

ACKOFF, Russel L. **Pesquisa Operacional**. Rio de Janeiro: Livros Técnicos e Científicos, 1977

ADAMOPOULOS, George I.; PAPPIS, Costas P. Scheduling under a common due-date on parallel unrelated machines. **European Journal of Operational Research**, Amsterdã, v. 105, n. 3, p. 494-501, 1998.

ARAGÃO, Marcus Poggi de; PALMEIRA, Márcio de Moraes. **Tabu search and strong cutting planes for the flow shop scheduling problem**. 1996. Trabalho apresentado no Symposium on Combinatorial Optimization (CO96). Londres, 1996. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.139.1619> . Acesso em 3 maio 2011.

BALCI, R. G. Sargent. A methodology for cost-risk analysis in the statistical validation of simulation models. **Communications of ACM,** New York**,** v. 24, n. 4, p. 190-197, abr.1981.

CAMPELO, Ruy Eduardo; MACULAN, Nelson. **Algoritmos e heurísticas.** Rio de Janeiro: Editora da Universidade Federal Fluminense, 1994.

CHEN, Bo. Analysis of classes of heuristics for scheduling a two-stage flow shop with parallel machines at on stage. **Journal of the Operational Research Society**, Basingstoke, v. 46, p. 234-244, fev.1995.

CHENG, T.C.E. A heuristic for common due-date assignment and job scheduling on parallel machines. **Journal of the Operational Research Society**, Basingstoke, v. 40, n. 12, 1129-1135, dez.1989.

DESSOUKY, Maged M.; DESSOUKY, Mohamed I.; VERMA, Sushil K. Flowshop scheduling with identical jobs and uniform parallel machines. **European Journal of Operational Research**, Amsterdã, v. 109, n. 3, p. 630-631, set. 1998.

DREES, Lawrence David; WILHELM, Wilbert E. Scheduling experiments on a nuclear reactor using mixed integer programming. **Computers & Operations Research**, Amsterdã, v.28, n. 10, p. 1013-1037, set. 2001.

FLOOD, Robert L.; CARSON, Ewart R. **Dealing with complexity:** an introduction to the theory and application of systems science. 2. Ed. New York: Plenum Press, 1993.

GASS, S.I. Decision-aiding models: validation, assessment, and related issues for policy analysis. **Operations Research**, Hanover, v. 31, n. 4, p. 603-631, jul./ago.1983.

GENOULAZ, Valérie Botta. Hybrid flow shop scheduling with precedence constraints and time lags do minimize maximum lateness. **International Journal of Production Economics**, Amsterdã, v. 64, n. 1-3, p. 101-111, mar. 2000.

GERASOULIS A.; YANG, T. Efficient algorithms and a software tool for scheduling parallel computation. In: CHRÉTIENNE, Philippe [et al]. **Scheduling theory and its applications.** Chichester: Wiley, 1997, p. 111-142.

GLOVER, Fred. Tabu Search - Part I. **Operations Research Society of America - Journal on Computing**, Linthicum, Maryland, v. 1, n. 3, p. 190-206, fev.1989.

GLOVER, Fred. Tabu Search: part II. **Operations Research Society of America - Journal on Computing**, Linthicum, Maryland, v. 2, n. 1, p. 4-32, nov. 1990.

HOLANDA, Aurélio Buarque de. **Novo Dicionário Aurélio da Língua Portuguesa**. 2. ed. Rio de Janeiro: Nova Fronteira, 1986.

JAFAR, M.; BAHILL, A.T. Interactive verification of knowledge-based systems. **IEEE Expert,** Washington, DC, v.8, n. 1, p. 25-32, fev.1993. Disponível em: < http://www.computer.org/portal/web/csdl/doi/10.1109/64.193052> . Acesso em 4 maio 2011.

LENSTRA, J. K.; SHMOYS, D. B. Computer near-optimal schedules. In: CHRÉTIENNE, Philippe [et al]. **Scheduling theory and its applications.** Chichester: Wiley, 1997, p. 1-13.

MISER, H. J. M.; QUADE, E. S. Validation. In: MISER, H. J. M.; QUADE, E. S. **Handbook of systems analysis:** craft issues and procedural choices, New York : Wiley, 1988, p. 527-565.

MOCCELLIN, João Vitor. A new heuristic method for the permutation flow shop scheduling problem. **Journal of the Operational Reseach Society**, New York, v. 46, p. 883-886, 1995.

MOCCELLIN, J. V.; NAGANO, M. S. Evaluating the perfomance of tabu searh procedures for flow shop sequencing. **Journal of the Operational Research Society**, New York, v. 49, p. 1296-1302, 1998.

MÜLLER, Felipe M. **Algoritmos heurísticos e exatos para solução do problema de sequenciamento em processadores paralelos**. Campinas: UNICAMP, 1993. 133 f. Tese (Doutorado), Programa de Pós-Graduação em Engenharia Elétrica, Departamento de Engenharia de Sistemas, Universidade Estadual de Campinas, Campinas, 1993.

NAGAR, Amit; HERAGU, Sunderesh S; HADDOCK, Jorge. A branch-and-bound approach for a two-machine flowshop scheduling problem. **Journal of the Operational Research Society**, New York, v. 46, n. 6, p. 721-734, jun.1995.

NEGENMAN, Ebbe G. Local search algorithms for the multiprocessor flow shop scheduling problem. **European Journal of Operational Research**, Amsterdã, v. 128, n. 1, p. 147-158, 2001.

NOWICKI, Eugeniusz; SMUTNICKI, Czeslaw. The flow shop with parallel machines: a tabu search approach. **European Journal of Operational Research**, Amsterdã, v. 106, n. 2-3, p. 226-253, abr.1998.

PIDD, Michael. **Modelagem empresarial: ferramentas para tomada de decisão.** Porto Alegre. Bookman, 1998.

PIDD, Michael. **Tools for thinking**: modelling in management science. Chichester : John Wiley & Sons, 1996.

PINEDO, Michael. **Scheduling:** theory, algorithms and systems**.** New Jersey: Prentice Hall, 1995.

PRADO, Vaner José do. **Avaliando a eficiência das lojas da ECT do Rio Grande do Sul**. Porto Alegre: UFRGS, 2000. 187 f. Dissertação (Mestrado em Administração) - Programa de Pós-Graduação em Administração da Universidade Federal do Rio Grande do Sul. Porto Alegre, 2000.

SIMON, H. A.; NEWELL, A. Heuristic problem solving: the next advance for operations research. **Operations Research**, Linthicum, v. 6, n. 1, p. 1-10, jan./fev.1958

SIMON, H. A. On how to decide what to do. **Bell Journal of Economics**, New York, v. 9, n. 3, p. 494-507, 1978.

TAHA, H. **Operations Research:** an introduction. New York: Mac Millan, 1982.

VALLS, Vicente; PEREZ, M. Angeles; QUINTANILLA, M. Sacramento. A tabu search approach to machine scheduling, **European Journal of Operational Research**, Amsterdã, v. 106, n. 2-3, p. 277-300, abr.1998.

WAGNER, Harvey M. **Pesquisa Operacional.** 2. ed. Rio de Janeiro: Prentice Hall do Brasil,1986.

WATHIER, Adair José. **Implementação distribuída da busca tabu para solução do problema de sequenciamento em processadores paralelos**. Santa Maria: UFSM, 1999. Trabalho de Conclusão de Curso de Graduação (Bacharelado em Informática). Universidade Federal de Santa Maria. Santa Maria. 1999.

WENG, Michael X.; LU, John; REN, Haiying. Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. **International Jounal of Production Economics**, Amsterdã, v. 70, n. 3, p. 215-226, abr. 2001.

ANEXO 1

Algoritmo Proposto por Müller (1993)

Etapa 1: ALOCAÇÃO INICIAL

Passo 1 - Encontre os *p* predecessores globais e os *p* sucessores globais de cada tarefa J_i .

Passo 2 - Escolha aleatoriamente *m* tarefas e insira uma em cada um dos *m* processadores. Inicialize as listas de vizinhos locais de cada tarefa em cada processador.

Passo 3 - Insira uma tarefa ainda não alocada no processador que produz o menor acréscimo no tempo de finalização atual. Se não há mais tarefas a alocar, PARE.

Após cada inserção atualize as listas de vizinhos locais.

Guarde a solução inicial encontrada como a solução incumbente (melhor solução encontrada até o momento) e vá para a FASE TABU.

Etapa 2: FASE TABU

Suponha que $\overline{M_i}$ é o processador mais carregado (com maior tempo de finalização). Considere:

$C_{M_i J_j}$ = tempo de finalização (tempo necessário para finalizar todas as tarefas) do processador M_i se a tarefa J_j é inserida nele.

$C_{\overline{M}_i J_j}$ = tempo de finalização do processador $\overline{M_i}$ se a tarefa J_j é retirada dele.

Passo 1 - Faça o contador de iterações $t = 0$.

Passo 2 - Para cada tarefa J_j pertencente ao processador mais carregado, encontre:

- o conjunto de processadores candidatos a receber a tarefa J_j , definido como:

$M(J_j) = \{M_i / M_i \neq \overline{M}_i \text{e} M_i$ inclua entre as suas tarefas pelo menos uma sucessora global ou uma predecessora global de $J_j\}$

- o menor tempo de finalização causado pela inserção de J_j em um dos processadores do conjunto $M(J_j)$, definido como:

$$C_{M_i^* J_j} = \min_{M_i \in M(J_i)} \{C_{M_i J_j}\}$$

- o tempo máximo de finalização entre M_i^* e $\overline{M}_i$ se a tarefa J_j fosse transferida de $\overline{M}_i$ para M_i^*, definido como:

$$T_{J_j} = \max\{C_{\overline{M}_i J_j}, C_{M_i^* J_j}\}$$

Passo 3 - Escolha a tarefa J_j^* a ser retirada de $\overline{M}_i$ como sendo $J_j^* = \text{argmin}\{T_{J_j}\}$ e J_j^* não tabu (a menos que o tempo de finalização produzido por este movimento seja melhor que o da solução incumbente - *CRITÉRIO DE ASPIRAÇÃO*).

Então, retire J_j^* de $\overline{M}_i$ e insira em M_i^*.

Proíba o movimento reverso por θ iterações.

Passo 4 - Atualize a lista das *q* tarefas predecessoras e sucessoras locais afetadas pelas

trocas nos processadores $\overline{M}_i$ e M_i^*.

Identifique o processador mais carregado e, se necessário, atualize a solução incumbente.

Passo 5 - Incremente o contador de iterações $t = t + 1$.

Se o número máximo de iterações sem melhoria da solução não foi alcançado, vá para o PASSO 2. Caso contrário vá para a FASE DE PÓS-OTIMIZAÇÃO.

Etapa 3: FASE DE PÓS-OTIMIZAÇÃO

Passo 1 - Considere M_i o processador mais carregado na solução incumbente e C_{M_i} o tempo de finalização deste processador.

Passo 2 - Para cada uma das tarefas alocadas a M_i realize uma retirada e uma reinserção no mesmo processador gerando um novo tempo de finalização $\overline{C}_{M_i}$.

Defina $C^*_{M_i} = \underset{J_j \in M_i}{MIN} \left\{ \overline{C}_{M_i} \right\}$ como sendo o tempo de finalização mínimo encontrado após as retiradas e reinserções.

Passo 3 - Se $C^*_{M_i} \geq C_{M_i}$, o tempo mínimo de finalização não se altera e PARE.

Se $C^*_{M_i} < C_{M_i}$, mas M_i ainda é o processador mais carregado, atualize a solução incumbente e PARE. Caso contrário, atualize a solução incumbente e vá para o PASSO 1.

ANEXO 2

Modelo de Programação Linear Inteira, para sequenciamento de 5 tarefas em 2 processadores, com 204 variáveis binárias.

O formato $T_t P_p O_o S_s$ apresentado no modelo de programação linear abaixo considera cada letra para substituir o índice, sendo, $A = 1, B = 2, C = 3, ... Z = 26$.

```
MIN
TTP01 + TTP02 + YT0102 + NT0102
ST
!Tempo Total de Cada Processador
! balanceando, variável auxiliar TTP * total do processador
1TAP1OASB + 1TAP1OASC + 1TAP1OASD + 1TAP1OASE
 + 1TAP1OBSB + 1TAP1OBSC + 1TAP1OBSD + 1TAP1OBSE
 + 1TAP1OCSB + 1TAP1OCSC + 1TAP1OCSD + 1TAP1OCSE
 + 1TAP1ODSB + 1TAP1ODSC + 1TAP1ODSD + 1TAP1ODSE
 + 1TAP1OESB + 1TAP1OESC + 1TAP1OESD + 1TAP1OESE
 + 3TBP1OASA + 3TBP1OASC + 3TBP1OASD + 3TBP1OASE
 + 3TBP1OBSA + 3TBP1OBSC + 3TBP1OBSD + 3TBP1OBSE
 + 3TBP1OCSA + 3TBP1OCSC + 3TBP1OCSD + 3TBP1OCSE
 + 3TBP1ODSA + 3TBP1ODSC + 3TBP1ODSD + 3TBP1ODSE
 + 3TBP1OESA + 3TBP1OESC + 3TBP1OESD + 3TBP1OESE
 + 2TCP1OASA + 2TCP1OASB + 2TCP1OASD + 2TCP1OASE
 + 2TCP1OBSA + 2TCP1OBSB + 2TCP1OBSD + 2TCP1OBSE
 + 2TCP1OCSA + 2TCP1OCSB + 2TCP1OCSD + 2TCP1OCSE
 + 2TCP1ODSA + 2TCP1ODSB + 2TCP1ODSD + 2TCP1ODSE
 + 2TCP1OESA + 2TCP1OESB + 2TCP1OESD + 2TCP1OESE
 + 2TDP1OASA + 2TDP1OASB + 2TDP1OASC + 2TDP1OASE
 + 2TDP1OBSA + 2TDP1OBSB + 2TDP1OBSC + 2TDP1OBSE
 + 2TDP1OCSA + 2TDP1OCSB + 2TDP1OCSC + 2TDP1OCSE
 + 2TDP1ODSA + 2TDP1ODSB + 2TDP1ODSC + 2TDP1ODSE
 + 2TDP1OESA + 2TDP1OESB + 2TDP1OESC + 2TDP1OESE
 + 1TEP1OASA + 1TEP1OASB + 1TEP1OASC + 1TEP1OASD
 + 1TEP1OBSA + 1TEP1OBSB + 1TEP1OBSC + 1TEP1OBSD
 + 1TEP1OCSA + 1TEP1OCSB + 1TEP1OCSC + 1TEP1OCSD
 + 1TEP1ODSA + 1TEP1ODSB + 1TEP1ODSC + 1TEP1ODSD
 + 1TEP1OESA + 1TEP1OESB + 1TEP1OESC + 1TEP1OESD
 - TTP01 =0
! balanceando, variavel auxiliar TTP * total do processador
 1TAP2OASB + 1TAP2OASC + 1TAP2OASD + 1TAP2OASE
```

+ 1TAP2OBSB + 1TAP2OBSC + 1TAP2OBSD + 1TAP2OBSE
+ 1TAP2OCSB + 1TAP2OCSC + 1TAP2OCSD + 1TAP2OCSE
+ 1TAP2ODSB + 1TAP2ODSC + 1TAP2ODSD + 1TAP2ODSE
+ 1TAP2OESB + 1TAP2OESC + 1TAP2OESD + 1TAP2OESE
+ 3TBP2OASA + 3TBP2OASC + 3TBP2OASD + 3TBP2OASE
+ 3TBP2OBSA + 3TBP2OBSC + 3TBP2OBSD + 3TBP2OBSE
+ 3TBP2OCSA + 3TBP2OCSC + 3TBP2OCSD + 3TBP2OCSE
+ 3TBP2ODSA + 3TBP2ODSC + 3TBP2ODSD + 3TBP2ODSE
+ 3TBP2OESA + 3TBP2OESC + 3TBP2OESD + 3TBP2OESE
+ 2TCP2OASA + 2TCP2OASB + 2TCP2OASD + 2TCP2OASE
+ 2TCP2OBSA + 2TCP2OBSB + 2TCP2OBSD + 2TCP2OBSE
+ 2TCP2OCSA + 2TCP2OCSB + 2TCP2OCSD + 2TCP2OCSE
+ 2TCP2ODSA + 2TCP2ODSB + 2TCP2ODSD + 2TCP2ODSE
+ 2TCP2OESA + 2TCP2OESB + 2TCP2OESD + 2TCP2OESE
+ 2TDP2OASA + 2TDP2OASB + 2TDP2OASC + 2TDP2OASE
+ 2TDP2OBSA + 2TDP2OBSB + 2TDP2OBSC + 2TDP2OBSE
+ 2TDP2OCSA + 2TDP2OCSB + 2TDP2OCSC + 2TDP2OCSE
+ 2TDP2ODSA + 2TDP2ODSB + 2TDP2ODSC + 2TDP2ODSE
+ 2TDP2OESA + 2TDP2OESB + 2TDP2OESC + 2TDP2OESE
+ 1TEP2OASA + 1TEP2OASB + 1TEP2OASC + 1TEP2OASD
+ 1TEP2OBSA + 1TEP2OBSB + 1TEP2OBSC + 1TEP2OBSD
+ 1TEP2OCSA + 1TEP2OCSB + 1TEP2OCSC + 1TEP2OCSD
+ 1TEP2ODSA + 1TEP2ODSB + 1TEP2ODSC + 1TEP2ODSD
+ 1TEP2OESA + 1TEP2OESB + 1TEP2OESC + 1TEP2OESD
- TTP02 =0
!para manter o equilibrio entre os processadores
! balanceando, variavel auxiliar YT??
1TAP1OASB - 1TAP2OASB + 1TAP1OASC - 1TAP2OASC + 1TAP1OASD -
1TAP2OASD + 1TAP1OASE - 1TAP2OASE + 1TAP1OBSB - 1TAP2OBSB +
1TAP1OBSC - 1TAP2OBSC + 1TAP1OBSD - 1TAP2OBSD + 1TAP1OBSE -
1TAP2OBSE + 1TAP1OCSB - 1TAP2OCSB + 1TAP1OCSC - 1TAP2OCSC +
1TAP1OCSD - 1TAP2OCSD + 1TAP1OCSE - 1TAP2OCSE + 1TAP1ODSB -
1TAP2ODSB + 1TAP1ODSC - 1TAP2ODSC + 1TAP1ODSD - 1TAP2ODSD +
1TAP1ODSE - 1TAP2ODSE + 1TAP1OESB - 1TAP2OESB + 1TAP1OESC -
1TAP2OESC + 1TAP1OESD - 1TAP2OESD + 1TAP1OESE - 1TAP2OESE
+ 3TBP1OASA - 3TBP2OASA + 3TBP1OASC - 3TBP2OASC + 3TBP1OASD
- 3TBP2OASD + 3TBP1OASE - 3TBP2OASE + 3TBP1OBSA - 3TBP2OBSA
+ 3TBP1OBSC - 3TBP2OBSC + 3TBP1OBSD - 3TBP2OBSD + 3TBP1OBSE
- 3TBP2OBSE + 3TBP1OCSA - 3TBP2OCSA + 3TBP1OCSC - 3TBP2OCSC
+ 3TBP1OCSD - 3TBP2OCSD + 3TBP1OCSE - 3TBP2OCSE + 3TBP1ODSA
- 3TBP2ODSA + 3TBP1ODSC - 3TBP2ODSC + 3TBP1ODSD - 3TBP2ODSD
+ 3TBP1ODSE - 3TBP2ODSE + 3TBP1OESA - 3TBP2OESA + 3TBP1OESC
- 3TBP2OESC + 3TBP1OESD - 3TBP2OESD + 3TBP1OESE - 3TBP2OESE
+ 2TCP1OASA - 2TCP2OASA + 2TCP1OASB - 2TCP2OASB + 2TCP1OASD
- 2TCP2OASD + 2TCP1OASE - 2TCP2OASE + 2TCP1OBSA - 2TCP2OBSA
+ 2TCP1OBSB - 2TCP2OBSB + 2TCP1OBSD - 2TCP2OBSD + 2TCP1OBSE
- 2TCP2OBSE + 2TCP1OCSA - 2TCP2OCSA + 2TCP1OCSB - 2TCP2OCSB
+ 2TCP1OCSD - 2TCP2OCSD + 2TCP1OCSE - 2TCP2OCSE + 2TCP1ODSA
- 2TCP2ODSA + 2TCP1ODSB - 2TCP2ODSB + 2TCP1ODSD - 2TCP2ODSD
+ 2TCP1ODSE - 2TCP2ODSE + 2TCP1OESA - 2TCP2OESA + 2TCP1OESB

- 2TCP2OESB + 2TCP1OESD - 2TCP2OESD + 2TCP1OESE - 2TCP2OESE
+ 2TDP1OASA - 2TDP2OASA + 2TDP1OASB - 2TDP2OASB + 2TDP1OASC
- 2TDP2OASC + 2TDP1OASE - 2TDP2OASE + 2TDP1OBSA - 2TDP2OBSA
+ 2TDP1OBSB - 2TDP2OBSB + 2TDP1OBSC - 2TDP2OBSC + 2TDP1OBSE
- 2TDP2OBSE + 2TDP1OCSA - 2TDP2OCSA + 2TDP1OCSB - 2TDP2OCSB
+ 2TDP1OCSC - 2TDP2OCSC + 2TDP1OCSE - 2TDP2OCSE + 2TDP1ODSA
- 2TDP2ODSA + 2TDP1ODSB - 2TDP2ODSB + 2TDP1ODSC - 2TDP2ODSC
+ 2TDP1ODSE - 2TDP2ODSE + 2TDP1OESA - 2TDP2OESA + 2TDP1OESB
- 2TDP2OESB + 2TDP1OESC - 2TDP2OESC + 2TDP1OESE - 2TDP2OESE
+ 1TEP1OASA - 1TEP2OASA + 1TEP1OASB - 1TEP2OASB + 1TEP1OASC
- 1TEP2OASC + 1TEP1OASD - 1TEP2OASD + 1TEP1OBSA - 1TEP2OBSA
+ 1TEP1OBSB - 1TEP2OBSB + 1TEP1OBSC - 1TEP2OBSC + 1TEP1OBSD
- 1TEP2OBSD + 1TEP1OCSA - 1TEP2OCSA + 1TEP1OCSB - 1TEP2OCSB
+ 1TEP1OCSC - 1TEP2OCSC + 1TEP1OCSD - 1TEP2OCSD + 1TEP1ODSA
- 1TEP2ODSA + 1TEP1ODSB - 1TEP2ODSB + 1TEP1ODSC - 1TEP2ODSC
+ 1TEP1ODSD - 1TEP2ODSD + 1TEP1OESA - 1TEP2OESA + 1TEP1OESB
- 1TEP2OESB + 1TEP1OESC - 1TEP2OESC + 1TEP1OESD - 1TEP2OESD
+ YT0102>=0
! balanceando, variavel auxiliar NT??
1TAP1OASB - 1TAP2OASB + 1TAP1OASC - 1TAP2OASC + 1TAP1OASD -
1TAP2OASD + 1TAP1OASE - 1TAP2OASE + 1TAP1OBSB - 1TAP2OBSB +
1TAP1OBSC - 1TAP2OBSC + 1TAP1OBSD - 1TAP2OBSD + 1TAP1OBSE -
1TAP2OBSE + 1TAP1OCSB - 1TAP2OCSB + 1TAP1OCSC - 1TAP2OCSC +
1TAP1OCSD - 1TAP2OCSD + 1TAP1OCSE - 1TAP2OCSE + 1TAP1ODSB -
1TAP2ODSB + 1TAP1ODSC - 1TAP2ODSC + 1TAP1ODSD - 1TAP2ODSD +
1TAP1ODSE - 1TAP2ODSE + 1TAP1OESB - 1TAP2OESB + 1TAP1OESC -
1TAP2OESC + 1TAP1OESD - 1TAP2OESD + 1TAP1OESE - 1TAP2OESE
+ 3TBP1OASA - 3TBP2OASA + 3TBP1OASC - 3TBP2OASC + 3TBP1OASD
- 3TBP2OASD + 3TBP1OASE - 3TBP2OASE + 3TBP1OBSA - 3TBP2OBSA
+ 3TBP1OBSC - 3TBP2OBSC + 3TBP1OBSD - 3TBP2OBSD + 3TBP1OBSE
- 3TBP2OBSE + 3TBP1OCSA - 3TBP2OCSA + 3TBP1OCSC - 3TBP2OCSC
+ 3TBP1OCSD - 3TBP2OCSD + 3TBP1OCSE - 3TBP2OCSE + 3TBP1ODSA
- 3TBP2ODSA + 3TBP1ODSC - 3TBP2ODSC + 3TBP1ODSD - 3TBP2ODSD
+ 3TBP1ODSE - 3TBP2ODSE + 3TBP1OESA - 3TBP2OESA + 3TBP1OESC
- 3TBP2OESC + 3TBP1OESD - 3TBP2OESD + 3TBP1OESE - 3TBP2OESE
+ 2TCP1OASA - 2TCP2OASA + 2TCP1OASB - 2TCP2OASB + 2TCP1OASD
- 2TCP2OASD + 2TCP1OASE - 2TCP2OASE + 2TCP1OBSA - 2TCP2OBSA
+ 2TCP1OBSB - 2TCP2OBSB + 2TCP1OBSD - 2TCP2OBSD + 2TCP1OBSE
- 2TCP2OBSE + 2TCP1OCSA - 2TCP2OCSA + 2TCP1OCSB - 2TCP2OCSB
+ 2TCP1OCSD - 2TCP2OCSD + 2TCP1OCSE - 2TCP2OCSE + 2TCP1ODSA
- 2TCP2ODSA + 2TCP1ODSB - 2TCP2ODSB + 2TCP1ODSD - 2TCP2ODSD
+ 2TCP1ODSE - 2TCP2ODSE + 2TCP1OESA - 2TCP2OESA + 2TCP1OESB
- 2TCP2OESB + 2TCP1OESD - 2TCP2OESD + 2TCP1OESE - 2TCP2OESE
+ 2TDP1OASA - 2TDP2OASA + 2TDP1OASB - 2TDP2OASB + 2TDP1OASC
- 2TDP2OASC + 2TDP1OASE - 2TDP2OASE + 2TDP1OBSA - 2TDP2OBSA
+ 2TDP1OBSB - 2TDP2OBSB + 2TDP1OBSC - 2TDP2OBSC + 2TDP1OBSE
- 2TDP2OBSE + 2TDP1OCSA - 2TDP2OCSA + 2TDP1OCSB - 2TDP2OCSB
+ 2TDP1OCSC - 2TDP2OCSC + 2TDP1OCSE - 2TDP2OCSE + 2TDP1ODSA
- 2TDP2ODSA + 2TDP1ODSB - 2TDP2ODSB + 2TDP1ODSC - 2TDP2ODSC
+ 2TDP1ODSE - 2TDP2ODSE + 2TDP1OESA - 2TDP2OESA + 2TDP1OESB

- 2TDP2OESB + 2TDP1OESC - 2TDP2OESC + 2TDP1OESE - 2TDP2OESE
+ 1TEP1OASA - 1TEP2OASA + 1TEP1OASB - 1TEP2OASB + 1TEP1OASC
- 1TEP2OASC + 1TEP1OASD - 1TEP2OASD + 1TEP1OBSA - 1TEP2OBSA
+ 1TEP1OBSB - 1TEP2OBSB + 1TEP1OBSC - 1TEP2OBSC + 1TEP1OBSD
- 1TEP2OBSD + 1TEP1OCSA - 1TEP2OCSA + 1TEP1OCSB - 1TEP2OCSB
+ 1TEP1OCSC - 1TEP2OCSC + 1TEP1OCSD - 1TEP2OCSD + 1TEP1ODSA
- 1TEP2ODSA + 1TEP1ODSB - 1TEP2ODSB + 1TEP1ODSC - 1TEP2ODSC
+ 1TEP1ODSD - 1TEP2ODSD + 1TEP1OESA - 1TEP2OESA + 1TEP1OESB
- 1TEP2OESB + 1TEP1OESC - 1TEP2OESC + 1TEP1OESD - 1TEP2OESD
- NT0102<= 0
!Cada tarefa so pode ser executada uma vez em cada processador
TAP1OASB + TAP1OASC + TAP1OASD + TAP1OASE + TBP1OASA +
TBP1OASC + TBP1OASD + TBP1OASE + TCP1OASA + TCP1OASB +
TCP1OASD + TCP1OASE + TDP1OASA + TDP1OASB + TDP1OASC +
TDP1OASE + TEP1OASA + TEP1OASB + TEP1OASC + TEP1OASD <= 1
TAP1OBSB + TAP1OBSC + TAP1OBSD + TAP1OBSE + TBP1OBSA +
TBP1OBSC + TBP1OBSD + TBP1OBSE + TCP1OBSA + TCP1OBSB +
TCP1OBSD + TCP1OBSE + TDP1OBSA + TDP1OBSB + TDP1OBSC +
TDP1OBSE + TEP1OBSA + TEP1OBSB + TEP1OBSC + TEP1OBSD <= 1
TAP1OCSB + TAP1OCSC + TAP1OCSD + TAP1OCSE + TBP1OCSA +
TBP1OCSC + TBP1OCSD + TBP1OCSE + TCP1OCSA + TCP1OCSB +
TCP1OCSD + TCP1OCSE + TDP1OCSA + TDP1OCSB + TDP1OCSC +
TDP1OCSE + TEP1OCSA + TEP1OCSB + TEP1OCSC + TEP1OCSD <= 1
TAP1ODSB + TAP1ODSC + TAP1ODSD + TAP1ODSE + TBP1ODSA +
TBP1ODSC + TBP1ODSD + TBP1ODSE + TCP1ODSA + TCP1ODSB +
TCP1ODSD + TCP1ODSE + TDP1ODSA + TDP1ODSB + TDP1ODSC +
TDP1ODSE + TEP1ODSA + TEP1ODSB + TEP1ODSC + TEP1ODSD <= 1
TAP1OESB + TAP1OESC + TAP1OESD + TAP1OESE + TBP1OESA +
TBP1OESC + TBP1OESD + TBP1OESE + TCP1OESA + TCP1OESB +
TCP1OESD + TCP1OESE + TDP1OESA + TDP1OESB + TDP1OESC +
TDP1OESE + TEP1OESA + TEP1OESB + TEP1OESC + TEP1OESD <= 1
TAP2OASB + TAP2OASC + TAP2OASD + TAP2OASE + TBP2OASA +
TBP2OASC + TBP2OASD + TBP2OASE + TCP2OASA + TCP2OASB +
TCP2OASD + TCP2OASE + TDP2OASA + TDP2OASB + TDP2OASC +
TDP2OASE + TEP2OASA + TEP2OASB + TEP2OASC + TEP2OASD <= 1
TAP2OBSB + TAP2OBSC + TAP2OBSD + TAP2OBSE + TBP2OBSA +
TBP2OBSC + TBP2OBSD + TBP2OBSE + TCP2OBSA + TCP2OBSB +
TCP2OBSD + TCP2OBSE + TDP2OBSA + TDP2OBSB + TDP2OBSC +
TDP2OBSE + TEP2OBSA + TEP2OBSB + TEP2OBSC + TEP2OBSD <= 1
TAP2OCSB + TAP2OCSC + TAP2OCSD + TAP2OCSE + TBP2OCSA +
TBP2OCSC + TBP2OCSD + TBP2OCSE + TCP2OCSA + TCP2OCSB +
TCP2OCSD + TCP2OCSE + TDP2OCSA + TDP2OCSB + TDP2OCSC +
TDP2OCSE + TEP2OCSA + TEP2OCSB + TEP2OCSC + TEP2OCSD <= 1
TAP2ODSB + TAP2ODSC + TAP2ODSD + TAP2ODSE + TBP2ODSA +
TBP2ODSC + TBP2ODSD + TBP2ODSE + TCP2ODSA + TCP2ODSB +
TCP2ODSD + TCP2ODSE + TDP2ODSA + TDP2ODSB + TDP2ODSC +
TDP2ODSE + TEP2ODSA + TEP2ODSB + TEP2ODSC + TEP2ODSD <= 1
TAP2OESB + TAP2OESC + TAP2OESD + TAP2OESE + TBP2OESA +
TBP2OESC + TBP2OESD + TBP2OESE + TCP2OESA + TCP2OESB +
TCP2OESD + TCP2OESE + TDP2OESA + TDP2OESB + TDP2OESC +

TDP2OESE + TEP2OESA + TEP2OESB + TEP2OESC + TEP2OESD <= 1
!A tarefa com setup X deve ser precedida da tarefa X
TAP1OASB + TAP1OASC + TAP1OASD + TAP1OASE - TBP1OBSA -
TCP1OBSA - TDP1OBSA - TEP1OBSA >= 0
TAP1OBSB + TAP1OBSC + TAP1OBSD + TAP1OBSE - TBP1OCSA -
TCP1OCSA - TDP1OCSA - TEP1OCSA >= 0
TAP1OCSB + TAP1OCSC + TAP1OCSD + TAP1OCSE - TBP1ODSA -
TCP1ODSA - TDP1ODSA - TEP1ODSA >= 0
TAP1ODSB + TAP1ODSC + TAP1ODSD + TAP1ODSE - TBP1OESA -
TCP1OESA - TDP1OESA - TEP1OESA >= 0
TBP1OASA + TBP1OASC + TBP1OASD + TBP1OASE - TAP1OBSB -
TCP1OBSB - TDP1OBSB - TEP1OBSB >= 0
TBP1OBSA + TBP1OBSC + TBP1OBSD + TBP1OBSE - TAP1OCSB -
TCP1OCSB - TDP1OCSB - TEP1OCSB >= 0
TBP1OCSA + TBP1OCSC + TBP1OCSD + TBP1OCSE - TAP1ODSB -
TCP1ODSB - TDP1ODSB - TEP1ODSB >= 0
TBP1ODSA + TBP1ODSC + TBP1ODSD + TBP1ODSE - TAP1OESB -
TCP1OESB - TDP1OESB - TEP1OESB >= 0
TCP1OASA + TCP1OASB + TCP1OASD + TCP1OASE - TAP1OBSC -
TBP1OBSC - TDP1OBSC - TEP1OBSC >= 0
TCP1OBSA + TCP1OBSB + TCP1OBSD + TCP1OBSE - TAP1OCSC -
TBP1OCSC - TDP1OCSC - TEP1OCSC >= 0
TCP1OCSA + TCP1OCSB + TCP1OCSD + TCP1OCSE - TAP1ODSC -
TBP1ODSC - TDP1ODSC - TEP1ODSC >= 0
TCP1ODSA + TCP1ODSB + TCP1ODSD + TCP1ODSE - TAP1OESC -
TBP1OESC - TDP1OESC - TEP1OESC >= 0
TDP1OASA + TDP1OASB + TDP1OASC + TDP1OASE - TAP1OBSD -
TBP1OBSD - TCP1OBSD - TEP1OBSD >= 0
TDP1OBSA + TDP1OBSB + TDP1OBSC + TDP1OBSE - TAP1OCSD -
TBP1OCSD - TCP1OCSD - TEP1OCSD >= 0
TDP1OCSA + TDP1OCSB + TDP1OCSC + TDP1OCSE - TAP1ODSD -
TBP1ODSD - TCP1ODSD - TEP1ODSD >= 0
TDP1ODSA + TDP1ODSB + TDP1ODSC + TDP1ODSE - TAP1OESD -
TBP1OESD - TCP1OESD - TEP1OESD >= 0
TEP1OASA + TEP1OASB + TEP1OASC + TEP1OASD - TAP1OBSE -
TBP1OBSE - TCP1OBSE - TDP1OBSE >= 0
TEP1OBSA + TEP1OBSB + TEP1OBSC + TEP1OBSD - TAP1OCSE -
TBP1OCSE - TCP1OCSE - TDP1OCSE >= 0
TEP1OCSA + TEP1OCSB + TEP1OCSC + TEP1OCSD - TAP1ODSE -
TBP1ODSE - TCP1ODSE - TDP1ODSE >= 0
TEP1ODSA + TEP1ODSB + TEP1ODSC + TEP1ODSD - TAP1OESE -
TBP1OESE - TCP1OESE - TDP1OESE >= 0
TAP2OASB + TAP2OASC + TAP2OASD + TAP2OASE - TBP2OBSA -
TCP2OBSA - TDP2OBSA - TEP2OBSA >= 0
TAP2OBSB + TAP2OBSC + TAP2OBSD + TAP2OBSE - TBP2OCSA -
TCP2OCSA - TDP2OCSA - TEP2OCSA >= 0
TAP2OCSB + TAP2OCSC + TAP2OCSD + TAP2OCSE - TBP2ODSA -
TCP2ODSA - TDP2ODSA - TEP2ODSA >= 0
TAP2ODSB + TAP2ODSC + TAP2ODSD + TAP2ODSE - TBP2OESA -
TCP2OESA - TDP2OESA - TEP2OESA >= 0

TBP2OASA + TBP2OASC + TBP2OASD + TBP2OASE - TAP2OBSB -
TCP2OBSB - TDP2OBSB - TEP2OBSB >= 0
TBP2OBSA + TBP2OBSC + TBP2OBSD + TBP2OBSE - TAP2OCSB -
TCP2OCSB - TDP2OCSB - TEP2OCSB >= 0
TBP2OCSA + TBP2OCSC + TBP2OCSD + TBP2OCSE - TAP2ODSB -
TCP2ODSB - TDP2ODSB - TEP2ODSB >= 0
TBP2ODSA + TBP2ODSC + TBP2ODSD + TBP2ODSE - TAP2OESB -
TCP2OESB - TDP2OESB - TEP2OESB >= 0
TCP2OASA + TCP2OASB + TCP2OASD + TCP2OASE - TAP2OBSC -
TBP2OBSC - TDP2OBSC - TEP2OBSC >= 0
TCP2OBSA + TCP2OBSB + TCP2OBSD + TCP2OBSE - TAP2OCSC -
TBP2OCSC - TDP2OCSC - TEP2OCSC >= 0
TCP2OCSA + TCP2OCSB + TCP2OCSD + TCP2OCSE - TAP2ODSC -
TBP2ODSC - TDP2ODSC - TEP2ODSC >= 0
TCP2ODSA + TCP2ODSB + TCP2ODSD + TCP2ODSE - TAP2OESC -
TBP2OESC - TDP2OESC - TEP2OESC >= 0
TDP2OASA + TDP2OASB + TDP2OASC + TDP2OASE - TAP2OBSD -
TBP2OBSD - TCP2OBSD - TEP2OBSD >= 0
TDP2OBSA + TDP2OBSB + TDP2OBSC + TDP2OBSE - TAP2OCSD -
TBP2OCSD - TCP2OCSD - TEP2OCSD >= 0
TDP2OCSA + TDP2OCSB + TDP2OCSC + TDP2OCSE - TAP2ODSD -
TBP2ODSD - TCP2ODSD - TEP2ODSD >= 0
TDP2ODSA + TDP2ODSB + TDP2ODSC + TDP2ODSE - TAP2OESD -
TBP2OESD - TCP2OESD - TEP2OESD >= 0
TEP2OASA + TEP2OASB + TEP2OASC + TEP2OASD - TAP2OBSE -
TBP2OBSE - TCP2OBSE - TDP2OBSE >= 0
TEP2OBSA + TEP2OBSB + TEP2OBSC + TEP2OBSD - TAP2OCSE -
TBP2OCSE - TCP2OCSE - TDP2OCSE >= 0
TEP2OCSA + TEP2OCSB + TEP2OCSC + TEP2OCSD - TAP2ODSE -
TBP2ODSE - TCP2ODSE - TDP2ODSE >= 0
TEP2ODSA + TEP2ODSB + TEP2ODSC + TEP2ODSD - TAP2OESE -
TBP2OESE - TCP2OESE - TDP2OESE >= 0
! controle de ordem de operacao
TAP1OASB + TAP1OASC + TAP1OASD + TAP1OASE + TBP1OASA +
TBP1OASC + TBP1OASD + TBP1OASE + TCP1OASA + TCP1OASB +
TCP1OASD + TCP1OASE + TDP1OASA + TDP1OASB + TDP1OASC +
TDP1OASE + TEP1OASA + TEP1OASB + TEP1OASC +
TEP1OASD-TAP1OBSB - TAP1OBSC - TAP1OBSD - TAP1OBSE -
TBP1OBSA - TBP1OBSC - TBP1OBSD - TBP1OBSE - TCP1OBSA -
TCP1OBSB - TCP1OBSD - TCP1OBSE - TDP1OBSA - TDP1OBSB -
TDP1OBSC - TDP1OBSE - TEP1OBSA - TEP1OBSB - TEP1OBSC -
TEP1OBSD >= 0
! controle de ordem de operacao
TAP2OASB + TAP2OASC + TAP2OASD + TAP2OASE + TBP2OASA +
TBP2OASC + TBP2OASD + TBP2OASE + TCP2OASA + TCP2OASB +
TCP2OASD + TCP2OASE + TDP2OASA + TDP2OASB + TDP2OASC +
TDP2OASE + TEP2OASA + TEP2OASB + TEP2OASC +
TEP2OASD-TAP2OBSB - TAP2OBSC - TAP2OBSD - TAP2OBSE -
TBP2OBSA - TBP2OBSC - TBP2OBSD - TBP2OBSE - TCP2OBSA -
TCP2OBSB - TCP2OBSD - TCP2OBSE - TDP2OBSA - TDP2OBSB -

TDP2OBSC - TDP2OBSE - TEP2OBSA - TEP2OBSB - TEP2OBSC -
TEP2OBSD >= 0
! controle de ordem de operação
TAP1OBSB + TAP1OBSC + TAP1OBSD + TAP1OBSE + TBP1OBSA +
TBP1OBSC + TBP1OBSD + TBP1OBSE + TCP1OBSA + TCP1OBSB +
TCP1OBSD + TCP1OBSE + TDP1OBSA + TDP1OBSB + TDP1OBSC +
TDP1OBSE + TEP1OBSA + TEP1OBSB + TEP1OBSC +
TEP1OBSD-TAP1OCSB - TAP1OCSC - TAP1OCSD - TAP1OCSE -
TBP1OCSA - TBP1OCSC - TBP1OCSD - TBP1OCSE - TCP1OCSA -
TCP1OCSB - TCP1OCSD - TCP1OCSE - TDP1OCSA - TDP1OCSB -
TDP1OCSC - TDP1OCSE - TEP1OCSA - TEP1OCSB - TEP1OCSC -
TEP1OCSD >= 0
! controle de ordem de operação
TAP2OBSB + TAP2OBSC + TAP2OBSD + TAP2OBSE + TBP2OBSA +
TBP2OBSC + TBP2OBSD + TBP2OBSE + TCP2OBSA + TCP2OBSB +
TCP2OBSD + TCP2OBSE + TDP2OBSA + TDP2OBSB + TDP2OBSC +
TDP2OBSE + TEP2OBSA + TEP2OBSB + TEP2OBSC +
TEP2OBSD-TAP2OCSB - TAP2OCSC - TAP2OCSD - TAP2OCSE -
TBP2OCSA - TBP2OCSC - TBP2OCSD - TBP2OCSE - TCP2OCSA -
TCP2OCSB - TCP2OCSD - TCP2OCSE - TDP2OCSA - TDP2OCSB -
TDP2OCSC - TDP2OCSE - TEP2OCSA - TEP2OCSB - TEP2OCSC -
TEP2OCSD >= 0
! controle de ordem de operação
TAP1OCSB + TAP1OCSC + TAP1OCSD + TAP1OCSE + TBP1OCSA +
TBP1OCSC + TBP1OCSD + TBP1OCSE + TCP1OCSA + TCP1OCSB +
TCP1OCSD + TCP1OCSE + TDP1OCSA + TDP1OCSB + TDP1OCSC +
TDP1OCSE + TEP1OCSA + TEP1OCSB + TEP1OCSC +
TEP1OCSD-TAP1ODSB - TAP1ODSC - TAP1ODSD - TAP1ODSE -
TBP1ODSA - TBP1ODSC - TBP1ODSD - TBP1ODSE - TCP1ODSA -
TCP1ODSB - TCP1ODSD - TCP1ODSE - TDP1ODSA - TDP1ODSB -
TDP1ODSC - TDP1ODSE - TEP1ODSA - TEP1ODSB - TEP1ODSC -
TEP1ODSD >= 0
! controle de ordem de operação
TAP2OCSB + TAP2OCSC + TAP2OCSD + TAP2OCSE + TBP2OCSA +
TBP2OCSC + TBP2OCSD + TBP2OCSE + TCP2OCSA + TCP2OCSB +
TCP2OCSD + TCP2OCSE + TDP2OCSA + TDP2OCSB + TDP2OCSC +
TDP2OCSE + TEP2OCSA + TEP2OCSB + TEP2OCSC +
TEP2OCSD-TAP2ODSB - TAP2ODSC - TAP2ODSD - TAP2ODSE -
TBP2ODSA - TBP2ODSC - TBP2ODSD - TBP2ODSE - TCP2ODSA -
TCP2ODSB - TCP2ODSD - TCP2ODSE - TDP2ODSA - TDP2ODSB -
TDP2ODSC - TDP2ODSE - TEP2ODSA - TEP2ODSB - TEP2ODSC -
TEP2ODSD >= 0
! controle de ordem de operação
TAP1ODSB + TAP1ODSC + TAP1ODSD + TAP1ODSE + TBP1ODSA +
TBP1ODSC + TBP1ODSD + TBP1ODSE + TCP1ODSA + TCP1ODSB +
TCP1ODSD + TCP1ODSE + TDP1ODSA + TDP1ODSB + TDP1ODSC +
TDP1ODSE + TEP1ODSA + TEP1ODSB + TEP1ODSC +
TEP1ODSD-TAP1OESB - TAP1OESC - TAP1OESD - TAP1OESE -
TBP1OESA - TBP1OESC - TBP1OESD - TBP1OESE - TCP1OESA -
TCP1OESB - TCP1OESD - TCP1OESE - TDP1OESA - TDP1OESB -

TDP1OESC - TDP1OESE - TEP1OESA - TEP1OESB - TEP1OESC -
TEP1OESD >= 0
! controle de ordem de operação
TAP2ODSB + TAP2ODSC + TAP2ODSD + TAP2ODSE + TBP2ODSA +
TBP2ODSC + TBP2ODSD + TBP2ODSE + TCP2ODSA + TCP2ODSB +
TCP2ODSD + TCP2ODSE + TDP2ODSA + TDP2ODSB + TDP2ODSC +
TDP2ODSE + TEP2ODSA + TEP2ODSB + TEP2ODSC +
TEP2ODSD-TAP2OESB - TAP2OESC - TAP2OESD - TAP2OESE -
TBP2OESA - TBP2OESC - TBP2OESD - TBP2OESE - TCP2OESA -
TCP2OESB - TCP2OESD - TCP2OESE - TDP2OESA - TDP2OESB -
TDP2OESC - TDP2OESE - TEP2OESA - TEP2OESB - TEP2OESC -
TEP2OESD >= 0
!cada tarefa será processada 1 vez
TAP1OASB + TAP1OASC + TAP1OASD + TAP1OASE + TAP1OBSB +
TAP1OBSC + TAP1OBSD + TAP1OBSE + TAP1OCSB + TAP1OCSC +
TAP1OCSD + TAP1OCSE + TAP1ODSB + TAP1ODSC + TAP1ODSD +
TAP1ODSE + TAP1OESB + TAP1OESC + TAP1OESD + TAP1OESE +
TAP2OASB + TAP2OASC + TAP2OASD + TAP2OASE + TAP2OBSB +
TAP2OBSC + TAP2OBSD + TAP2OBSE + TAP2OCSB + TAP2OCSC +
TAP2OCSD + TAP2OCSE + TAP2ODSB + TAP2ODSC + TAP2ODSD +
TAP2ODSE + TAP2OESB + TAP2OESC + TAP2OESD + TAP2OESE = 1
TBP1OASA + TBP1OASC + TBP1OASD + TBP1OASE + TBP1OBSA +
TBP1OBSC + TBP1OBSD + TBP1OBSE + TBP1OCSA + TBP1OCSC +
TBP1OCSD + TBP1OCSE + TBP1ODSA + TBP1ODSC + TBP1ODSD +
TBP1ODSE + TBP1OESA + TBP1OESC + TBP1OESD + TBP1OESE +
TBP2OASA + TBP2OASC + TBP2OASD + TBP2OASE + TBP2OBSA +
TBP2OBSC + TBP2OBSD + TBP2OBSE + TBP2OCSA + TBP2OCSC +
TBP2OCSD + TBP2OCSE + TBP2ODSA + TBP2ODSC + TBP2ODSD +
TBP2ODSE + TBP2OESA + TBP2OESC + TBP2OESD + TBP2OESE = 1
TCP1OASA + TCP1OASB + TCP1OASD + TCP1OASE + TCP1OBSA +
TCP1OBSB + TCP1OBSD + TCP1OBSE + TCP1OCSA + TCP1OCSB +
TCP1OCSD + TCP1OCSE + TCP1ODSA + TCP1ODSB + TCP1ODSD +
TCP1ODSE + TCP1OESA + TCP1OESB + TCP1OESD + TCP1OESE +
TCP2OASA + TCP2OASB + TCP2OASD + TCP2OASE + TCP2OBSA +
TCP2OBSB + TCP2OBSD + TCP2OBSE + TCP2OCSA + TCP2OCSB +
TCP2OCSD + TCP2OCSE + TCP2ODSA + TCP2ODSB + TCP2ODSD +
TCP2ODSE + TCP2OESA + TCP2OESB + TCP2OESD + TCP2OESE = 1
TDP1OASA + TDP1OASB + TDP1OASC + TDP1OASE + TDP1OBSA +
TDP1OBSB + TDP1OBSC + TDP1OBSE + TDP1OCSA + TDP1OCSB +
TDP1OCSC + TDP1OCSE + TDP1ODSA + TDP1ODSB + TDP1ODSC +
TDP1ODSE + TDP1OESA + TDP1OESB + TDP1OESC + TDP1OESE +
TDP2OASA + TDP2OASB + TDP2OASC + TDP2OASE + TDP2OBSA +
TDP2OBSB + TDP2OBSC + TDP2OBSE + TDP2OCSA + TDP2OCSB +
TDP2OCSC + TDP2OCSE + TDP2ODSA + TDP2ODSB + TDP2ODSC +
TDP2ODSE + TDP2OESA + TDP2OESB + TDP2OESC + TDP2OESE = 1
TEP1OASA + TEP1OASB + TEP1OASC + TEP1OASD + TEP1OBSA +
TEP1OBSB + TEP1OBSC + TEP1OBSD + TEP1OCSA + TEP1OCSB +
TEP1OCSC + TEP1OCSD + TEP1ODSA + TEP1ODSB + TEP1ODSC +
TEP1ODSD + TEP1OESA + TEP1OESB + TEP1OESC + TEP1OESD +
TEP2OASA + TEP2OASB + TEP2OASC + TEP2OASD + TEP2OBSA +

TEP2OBSB + TEP2OBSC + TEP2OBSD + TEP2OCSA + TEP2OCSB +
TEP2OCSC + TEP2OCSD + TEP2ODSA + TEP2ODSB + TEP2ODSC +
TEP2ODSD + TEP2OESA + TEP2OESB + TEP2OESC + TEP2OESD = 1
END
GIN 204

ANEXO 3

Posição de cada processador em cada situação analisada

Posição	Proc	Makespan
1	001	78
	002	72
	003	92
	004	64
	005	71
	006	66
	007	94
	008	79
	009	100
	010	107
	011	77
	012	78
2	001	77
	002	72
	003	92
	004	64
	005	71
	006	66
	007	92
	008	78
	009	100
	010	106
	011	77
	012	77
3	001	77
	002	72
	003	92
	004	64
	005	71
	006	66
	007	92
	008	78
	009	100
	010	106
	011	77
	012	77
4	001	77
	002	72
	003	92
	004	64
	005	71

Posição	Proc	*Makespan*
13	001	75
	002	63
	003	92
	004	51
	005	72
	006	67
	007	94
	008	73
	009	7
	010	100
	011	96
	012	72
	013	70
14	001	79
	002	63
	003	92
	004	51
	005	72
	006	67
	007	94
	008	73
	009	7
	010	100
	011	96
	012	72
	013	70
15	001	79
	002	63
	003	92
	004	51
	005	72
	006	67
	007	95
	008	73
	009	7
	010	100
	011	96
	012	72
	013	70
16	001	80
	002	63

Posição	Proc	*Makespan*
24	001	73
	002	79
	003	78
	004	42
	005	67
	006	63
	007	93
	008	88
	009	7
	010	98
	011	92
	012	66
	013	67
25	001	73
	002	79
	003	75
	004	42
	005	67
	006	63
	007	93
	008	85
	009	7
	010	95
	011	89
	012	66
	013	64
26	001	73
	002	79
	003	75
	004	42
	005	67
	006	63
	007	93
	008	85
	009	7
	010	95
	011	89
	012	66
	013	64
27	001	73
	002	79

	006	66
	007	92
	008	78
	009	100
	010	106
	011	77
	012	77
5	001	71
	002	69
	003	92
	004	64
	005	68
	006	66
	007	90
	008	75
	009	99
	010	100
	011	75
	012	70
6	001	71
	002	69
	003	92
	004	51
	005	75
	006	66
	007	98
	008	75
	009	99
	010	100
	011	75
	012	70
7	001	71
	002	69
	003	92
	004	51
	005	75
	006	66
	007	98
	008	75
	009	106
	010	93
	011	75
	012	70
8	001	71
	002	69
	003	92
	004	51
	005	75

	003	92
	004	52
	005	74
	006	70
	007	95
	008	74
	009	7
	010	100
	011	96
	012	74
	013	70
17	001	80
	002	63
	003	92
	004	52
	005	74
	006	70
	007	100
	008	74
	009	7
	010	100
	011	96
	012	74
	013	70
18	001	80
	002	63
	003	92
	004	52
	005	74
	006	70
	007	100
	008	74
	009	7
	010	100
	011	96
	012	74
	013	70
19	001	80
	002	63
	003	92
	004	52
	005	74
	006	70
	007	100
	008	74
	009	7
	010	100
	011	96

	003	75
	004	42
	005	67
	006	64
	007	93
	008	85
	009	7
	010	95
	011	89
	012	66
	013	64
28	001	73
	002	79
	003	75
	004	42
	005	67
	006	64
	007	93
	008	85
	009	7
	010	85
	011	89
	012	66
	013	64
29	001	72
	002	79
	003	75
	004	42
	005	67
	006	64
	007	93
	008	85
	009	7
	010	85
	011	89
	012	66
	013	64
30	001	72
	002	79
	003	75
	004	42
	005	67
	006	64
	007	93
	008	85
	009	7
	010	85
	011	89

	006	66
	007	98
	008	75
	009	106
	010	93
	011	75
	012	70
9	001	71
	002	69
	003	92
	004	51
	005	75
	006	66
	007	98
	008	76
	009	7
	010	106
	011	93
	012	75
	013	70
10	001	75
	002	63
	003	92
	004	51
	005	72
	006	67
	007	94
	008	73
	009	7
	010	101
	011	96
	012	72
	013	70
11	001	75
	002	63
	003	92
	004	51
	005	72
	006	67
	007	94
	008	73
	009	7
	010	100
	011	96
	012	72
	013	70
12	001	75
	002	63

	012	74
	013	70
20	001	79
	002	63
	003	78
	004	52
	005	73
	006	70
	007	100
	008	91
	009	7
	010	100
	011	95
	012	73
	013	70
21	001	79
	002	79
	003	78
	004	52
	005	73
	006	70
	007	100
	008	91
	009	7
	010	100
	011	95
	012	73
	013	70
22	001	73
	002	79
	003	78
	004	42
	005	67
	006	63
	007	93
	008	88
	009	7
	010	98
	011	89
	012	66
	013	67
23	001	73
	002	79
	003	78
	004	42
	005	67
	006	63
	007	93

	012	66
	013	64
31	001	72
	002	79
	003	75
	004	42
	005	67
	006	65
	007	93
	008	85
	009	7
	010	85
	011	89
	012	66
	013	64
32	001	85
	002	79
	003	79
	004	54
	005	91
	006	66
	007	98
	008	88
	009	7
	010	106
	011	105
	012	100
	013	63
33	001	81
	002	74
	003	113
	004	43
	005	87
	006	60
	007	91
	008	82
	009	7
	010	115
	011	99
	012	93
	013	29

	003	92
	004	51
	005	72
	006	67
	007	94
	008	73
	009	7
	010	100
	011	96
	012	72
	013	70

	008	88
	009	7
	010	98
	011	92
	012	66
	013	67