PRODUÇÃO

Algoritmo para o problema de seqüenciamento em máquinas paralelas não-relacionadas

Felipe Martins Müller
Doutor do Programa de Pós-Graduação
em Engenharia de Produção – Centro de Tecnologia
Universidade Federal de Santa Maria – UFSM
E-mail: felipe@inf.ufsm.br

Odon Bastos Dias
Mestre do Programa de Pós-Graduação
em Engenharia de Produção
Universidade Federal de Santa Maria – UFSM

Olinto César Bassi de Araújo
Mestre do Colégio Agrícola de Santa Maria
Universidade Federal de Santa Maria – UFSM
E-mail: olinto@casm.ufsm.br

Resumo

Este trabalho trata do problema de seqüenciamento de *n* tarefas independentes em *m* máquinas paralelas não-relacionadas com o objetivo de minimizar o tempo de execução da máquina mais carregada (*makespan*). É proposto um novo algoritmo de busca local em conexão com um esquema de vizinhança que usa estrutura de intervalos e o conceito de eficiência das máquinas para cada tarefa. O algoritmo proposto, denominado *Mutat*, é comparado com outros algoritmos para avaliar a qualidade das soluções obtidas. A nova abordagem encontra soluções que superam, em qualidade e tempo computacional, o melhor algoritmo de busca local encontrado na literatura para este problema.

Palavras-chave

Seqüenciamento, Máquinas não-relacionadas, Busca local, Heurísticas.

Algorithm to solve the unrelated parallel machine scheduling problem

Abstract

This work deals with the problem of scheduling n *independent jobs on* m *unrelated parallel machines with the objective of minimizing the makespan (the total elapsed time from the start of execution until all jobs are completed). In this work we propose a new local search algorithm in connection with a powerful neighborhood scheme that uses a structure of intervals and uses the efficiency of the machines for each job. The proposed algorithm, called* Mutat*, is compared with other algorithms in order to evaluate the quality of the solutions obtained. The new approach finds solutions that overcome, in quality and computational time, the best algorithm of local search found in the literature for this problem.*

Key words

Scheduling, Unrelated machines, Local search, Heuristics.

INTRODUÇÃO

O problema de seqüenciamento (*scheduling*) em máquinas paralelas, abordado neste artigo, é um dos problemas de otimização mais abordados na literatura especializada. Neste problema, o objetivo principal é alocar um determinado número (n) de tarefas independentes, com tempos de execução conhecidos, para um número (m) de máquinas paralelas. Após a distribuição das tarefas às máquinas, a soma dos tempos das tarefas pertencentes à máquina com a maior carga entre todas (*makespan*) deve ser a mínima possível.

Muitas vezes, um método originalmente proposto para realizar seqüenciamento em máquinas paralelas pode resolver problemas análogos em áreas diferentes. Como exemplo, pode-se imaginar uma fábrica com duas ou mais máquinas (processadores) em uma linha de produção. Qual a ordem de tarefas a serem executadas em cada máquina, considerando reduzir o seu tempo máximo de utilização? Como realizar a alocação das tarefas, se cada máquina possuir velocidade de processamento diferente? Ou ainda, se o tempo de execução de uma tarefa depende da máquina para a qual a tarefa é atribuída.

O problema de seqüenciamento em máquinas paralelas pode ser classificado quanto ao tipo de máquina utilizado: máquinas paralelas idênticas, máquinas paralelas uniformes e máquinas paralelas não-relacionadas (*unrelated*).

As máquinas paralelas são idênticas quando existe um conjunto único contendo os tempos de execução (ou finalização) das tarefas e estes tempos de execução permanecem constantes (idênticos) não importando para qual máquina uma tarefa é atribuída. Um exemplo prático acontece quando se decide implantar linhas de produção após exaustivos testes de simulação, isso faz com que os gargalos sejam identificados e a multiplicação da capacidade desses gargalos é feita pela colocação de várias máquinas em paralelo. Como esse processo é feito na implantação da linha de produção, é bastante provável que as máquinas sejam idênticas.

As máquinas paralelas são uniformes quando existe um conjunto único contendo os tempos de execução (ou finalização) das tarefas, mas os tempos de execução são alterados por um fator uniforme, dependendo da tarefa a ser atribuída a uma máquina ou à outra. O exemplo prático é o seguinte: para evitar gargalos devido a um aumento da produção, faz-se necessária a aquisição de mais máquinas em determinados pontos da produção, muitas vezes essas máquinas tem características idênticas, mas devido a melhorias tecnológicas têm velocidades de processamento diferentes.

As máquinas paralelas não estão relacionadas (*unrelated*) quando existem n tarefas para serem distribuídas entre as m máquinas, mas cada tarefa é representada por um subconjunto contendo m tempos de execução. Os valores pertencentes ao subconjunto podem ser diferentes e representam o tempo de execução da tarefa quando ela é atribuída a uma determinada máquina exclusivamente. Ou seja, o primeiro valor do subconjunto representa o tempo de execução da tarefa quando ela é atribuída à primeira máquina, o segundo valor representa o tempo de execução da tarefa quando ela é atribuída à segunda máquina, e assim sucessivamente até o m-ésimo valor. Isso acontece quando se fazem readequações do processo de fabricação, ou quando se têm células flexíveis de manufatura, onde várias máquinas estão aptas a desempenhar diversas tarefas, porém elas podem ser mais rápidas para determinados procedimentos enquanto outras são mais rápidas para outros, não havendo possibilidade de estabelecer uma relação de velocidade.

O Algoritmo *Mutat* possui quatro fases, uma fase construtiva e três fases de melhoramento.

Imaginando a possibilidade da divisão de uma tarefa entre duas máquinas, tem-se a chamada preempção. Em uma situação de não-preempção, uma tarefa uma vez alocada a uma máquina deve permanecer nela até o final de sua execução, sem interrupções. Os métodos para solução do problema de seqüenciamento em máquinas paralelas, apresentados a seguir, referem-se exclusivamente à situação de não-preempção. Como função, objetivo vai-se considerar a minimização do tempo máximo de finalização de todas as tarefas (*makespan*). Portanto, seguindo a classificação de três campos introduzida por (LAWLER *et al.*, 1989), os problemas de seqüenciamento em máquinas paralelas, sem preempção, com o objetivo de minimizar o *makespan*, passarão a ser denominados: $P||C_{max}$, para máquinas idênticas; $Q||C_{max}$, para máquina uniformes; e $R||C_{max}$, para máquinas não-relacionadas

Inicialmente, são abordados métodos existentes na literatura aplicados em máquinas paralelas idênticas e, após, os métodos para máquinas paralelas não-relacionadas, tema principal deste artigo.

MÉTODOS PARA MÁQUINAS PARALELAS IDÊNTICAS

Um dos resultados iniciais no estudo de alocação das tarefas em máquinas idênticas foi a definição de um limitan-

te para ser utilizado como referência, proposto por (MCNAUGHTON, 1959). O valor do limitante pode ser calculado somando-se o tempo de processamento das *n* tarefas e dividindo-o por *m*, o número de máquinas. Trata-se de uma média aritmética, representando a média de carga para cada máquina, possuindo as vantagens e desvantagens desta medida estatística.

Graham propôs (durante a década de 1960) algoritmos heurísticos com complexidade de tempo polinomial e teoremas demonstrando resultados. Seus algoritmos heurísticos são fáceis de implementar e muito rápidos na execução. Variações de seus algoritmos são encontradas com certa freqüência em adaptações para os problemas em máquinas paralelas uniformes e máquinas paralelas não-relacionadas.

O primeiro (GRAHAM, 1969) foi chamado de *LIST SCHEDULING* (*LS*). Trata-se de uma heurística construtiva, cuja representação algorítmica é tão pequena que por vezes alguns autores referem-se ao algoritmo como regra heurística. O enunciado de *LS* determina que as *n* tarefas sejam organizadas em uma lista, de ordem aleatória, sendo então alocadas, uma a uma, à máquina menos carregada. Cada vez que ocorrer empate na questão "máquina menos carregada", decide-se, arbitrariamente, para qual máquina atribuir a tarefa.

Após a aplicação da regra heurística *LS*, pode-se esperar uma solução onde a carga, na máquina com maior tempo de finalização, seja, no máximo, duas vezes maior que a solução exata (ou ótima).

Posteriormente, Graham melhorou o desempenho de pior caso ao introduzir o algoritmo *LPT* (*Longest Processing Time first*). A regra de distribuição foi alterada permitindo que as tarefas com maior tempo de execução sejam alocadas primeiro. Condições de impasse (empates) também são resolvidas arbitrariamente.

Formalmente, esta regra heurística (*LPT*) pode ser representada através do algoritmo:

Passo 1. Faça a ordenação decrescente das *n* tarefas.
Passo 2. Aplique a regra heurística *LS*.

Com suas raízes no problema de empacotamento (*bin packing*), a heurística construtiva *MULTIFIT* (COFFMAN *et al.*, 1978) demonstra que é possível adaptar um método para resolução de um problema de otimização existente (*bin packing*) para o problema de seqüenciamento.

As heurísticas *LS*, *LPT* e *MULTIFIT* vistas anteriormente, classificam-se como heurísticas construtivas. Representam um grupo de heurísticas muito influenciado pela filosofia de ordenação das tarefas.

A partir de agora, serão analisadas heurísticas classificadas como heurísticas de melhoramento, que introduziram a idéia de movimentos de realocação e trocas de tarefas entre máquinas, no final da década de 1970. A estratégia de realocação e troca possui uma importância muito grande neste artigo, pois o algoritmo proposto aqui representa um refinamento desta estratégia.

A primeira destas heurísticas de trocas foi apresentada por (FINN; HOROWITZ, 1979) e recebeu o nome de 0/1-*INTERCHANGE*. Apesar de existir a possibilidade de aplicar a heurística 0/1-*INTERCHANGE* após uma heurística de construção (*LS*, por exemplo), os autores determinaram que o algoritmo inicia alocando as *n* tarefas para as *m* máquinas aleatoriamente.

A etapa de realocação das tarefas entre máquinas começa com uma ordenação das cargas das máquinas, tal que $C_1 \geq C_2 \geq ... \geq C_m$, onde C_i representa a carga da *i*-ésima máquina. Na prática, é suficiente encontrar a máquina mais carregada (M_1) e a máquina menos carregada (M_m). A seguir, obtém-se a diferença entre os tempos de execução destas duas máquinas, $d = C_1 - C_m$. Se existir uma tarefa na máquina M_1, cujo tempo de execução seja menor que a diferença *d*, então esta tarefa é retirada de M_1 e transferida para a máquina menos carregada, M_m.

Cada realocação de tarefa provoca uma alteração na carga de duas máquinas, logo, uma nova ordenação das cargas ocorre, a diferença *d* é novamente calculada, e busca-se outra tarefa possível de ser transferida para a máquina M_m. A repetição deste processo segue enquanto existirem tarefas em M_1 com tempos de processamento menores que *d*.

Para o pior caso, a razão entre a solução heurística encontrada via aplicação de 0/1-*INTERCHANGE* e a solução exata do problema é 2. Isto representa que, na pior das hipóteses, a solução heurística será o dobro da solução exata. Porém, os autores afirmam que este valor diminui significativamente se o número de tarefas alocadas à máquina mais carregada M_1 for superior a 6, para qualquer *m*. Em tais circunstâncias, o valor da razão entre a solução heurística e a solução exata do problema será menor que aquele encontrado por *MULTIFIT*.

Foram propostas melhorias para o 0/1-*INTERCHANGE* (LANGSTON, 1982), passando a ser denominado *IMPROVED* 0/1-*INTERCHANGE*. Apesar de existirem muitas reformas potenciais para o sistema de realocações, as melhorias foram efetivadas no sistema de alocação inicial, procurando evitar alocações iniciais pobres, ocasionadas pela distribuição inadequada das tarefas com tempos de execução grandes, visto que o sistema de trocas não consegue movê-las para outras máquinas.

Alterações no sistema de realocações e trocas de tarefas, entre outras melhorias, foram realizadas por (MÜLLER, 1993), ao apresentar o algoritmo 3-FASES. Ao contrário dos outros algoritmos que adotam os movimentos de realocação e troca de tarefas como filosofia, este algo-

ritmo evita a ordenação inicial das tarefas. A primeira fase do algoritmo é uma etapa simples para alocação inicial das tarefas.

A segunda fase do algoritmo executa as realocações de tarefas, como também o faz a heurística 0/1-*INTERCHANGE*, porém, com a introdução de uma política de alvos baseada no limitante de McNAUGHTON, que beneficia primeiro a realocação das tarefas de tamanho menor. Somente na impossibilidade de ocorrerem estas realocações, o algoritmo passa a utilizar a diferença *d*, vista na heurística 0/1-*INTERCHANGE*.

A terceira fase representa a possibilidade de continuar realizando movimentos de troca a partir do ponto no qual o algoritmo 0/1-*INTERCHANGE* finaliza. Para ser possível realizar outras trocas além deste ponto, ainda utilizando a diferença *d*, o algoritmo busca duas tarefas, uma na máquina mais carregada e outra em alguma das máquinas, de modo que sua troca diminua o *makespan*.

Convém destacar que a troca de tarefas não é limitada somente entre a máquina menos carregada e a máquina mais carregada. As máquinas são ordenadas de acordo com suas cargas em ordem decrescente. Quando se esgotam as possibilidades de dupla troca entre os dois extremos, novas tentativas são realizadas envolvendo a máquina mais carregada e outra máquina, a partir da penúltima até a segunda, considerando a ordem decrescente de cargas, na esperança de reduzir ao mínimo o tempo de execução (*makespan*) da máquina mais carregada.

Quanto ao seu desempenho, de acordo com (FRANÇA *et al.*, 1994): "Este algoritmo apresenta desempenho muito bom, em relação à solução obtida, considerando-se a carga da máquina mais carregada (*makespan*), em comparação com os demais algoritmos anteriormente apresentados, e, ao mesmo tempo, em função da sua estrutura de intervalos, consegue um tempo de processamento semelhante aos demais algoritmos".

Exemplos numéricos de aplicação da heurística 3-FASES podem ser encontrados em (MÜLLER, 1993), para o problema de máquinas paralelas idênticas, e em (LIMBERGER, 1997) uma adaptação do algoritmo 3-FASES para o problema de máquinas paralelas uniformes.

Para o problema de máquinas paralelas uniformes, mais especificamente, podem ser encontradas adaptações do algoritmo *LIST SCHEDULING* em (LIU; LIU, 1974), (CHO; SAHNI, 1980). Variações do algoritmo *LPT* podem ser encontradas em (GONZALEZ *et al.*, 1977), (DOBSON, 1984) e (FRIESEN; LANGSTON, 1986). Uma extensão do algoritmo *MULTIFIT* pode ser encontrada em (FRIESEN; LANGSTON, 1983).

MÉTODOS PARA MÁQUINAS PARALELAS NÃO-RELACIONADAS

A partir de agora, vai-se discutir, especificamente, o problema $R||C_{max}$, que trata de alocar *n* tarefas independentes a *m* máquinas paralelas não-relacionadas, sendo que cada tarefa tem um tempo de processamento diferente para cada máquina. O objetivo é minimizar o tempo de execução da máquina mais carregada (*makespan*) (BAKER, 1974). Além disso, assume-se que todas as tarefas estão disponíveis para iniciar sua execução ao mesmo tempo, e uma máquina com mais de uma tarefa alocada a ela, deve executá-las uma após a outra, em alguma seqüência. Preempção não é permitida, ou seja, uma tarefa que inicia sua execução em uma máquina deve permanecer nela até o seu final.

A política de intervalos consiste em agrupar as tarefas em intervalos segundo seus tempos de processamento.

Para uma definição formal do problema $R||C_{max}$ considere um conjunto $J = \{J_1, J_2, ..., J_n\}$ de *n* tarefas que devem ser designadas a um conjunto $I = \{M_1, M_2, ..., M_m\}$ de *m* máquinas. Considere também $\{p_{ij}\}$ uma matriz *mxn* de tempos de processamento, onde p_{ij} é o tempo que a *i*-ésima máquina leva para executar a *j*-ésima tarefa e, $X = \{x_{ij}\}$, um conjunto de variáveis de decisão, onde $x_{ij} = 1$ caso a *j*-ésima tarefa seja designada para a máquina *i*, e $x_{ij} = 0$ caso contrário. O modelo matemático para $R||C_{max}$ pode ser visualizado pelas equações (1) a (4).

$$\min Z = c \qquad (1)$$

$$\text{s.a.} \quad \sum_{j=1}^{n} \left(p_{ij}\, x_{ij} \right) \le c, \quad i = 1, \ldots, m \qquad (2)$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n \qquad (3)$$

$$x_{ij} = 0 \text{ or } 1, \quad i = 1, \ldots, m \;;\; j = 1, \ldots, n \qquad (4)$$

O problema $R||C_{max}$ é considerado NP-Hard (GAREY; JOHNSON, 1979). Desta forma, acredita-se que a existência de um algoritmo com tempo polinomial para resolvê-lo na otimalidade é pouco provável. Apesar disso, encontram-se na literatura alguns exemplos de algoritmos exatos para o $R||C_{max}$, como em (HOROWITZ; SAHNI, 1976) e, mais

recentemente, (MARTELLO *et al.*, 1997). Ainda, encontram-se algoritmos exatos para o ITMAP (*Imbalanced Time Minimizing Assignment Problem*) em (MÜLLER; CAMOZZATO; ARAÚJO, 2001). O ITMAP pode ser considerado uma variação do $R||C_{max}$, diferenciando-se deste por conter a restrição adicional que cada máquina deve executar pelo menos uma tarefa. No entanto, estes algoritmos estão restritos à solução de instâncias de pequena dimensão.

Além do uso de intervalos alia-se o conceito de eficiência para melhor guiar a busca através do espaço de soluções.

Devido aos motivos apresentados, verifica-se que muito esforço de pesquisa tem sido feito para desenvolver algoritmos heurísticos que produzam soluções de boa qualidade em um tempo computacional razoável. (GLASS *et al*., 1994) apresentam um estudo comparativo de diferentes metaheurísticas para o $R||C_{max}$ (Busca Tabu, Algoritmos Genéticos e *Simulated Annealing*). (PIERSMA; VAN DIJK, 1996) apresentam um algoritmo de busca local que utiliza o conceito de eficiência (que será apresentado na seção 4) aliado a um grande número de ordenações intermediárias. Este algoritmo consegue soluções de qualidade comparável às obtidas pelas metaheurísticas estudadas em (GLASS *et al*., 1994).

Na próxima seção será proposto um algoritmo heurístico, denominado *Mutat*, para a resolução do $R||C_{max}$, que utiliza uma política de distribuição das tarefas em intervalos. Esta estrutura tem dois objetivos principais: eliminar os algoritmos de ordenação normalmente utilizados neste tipo de problema; e explorar de forma inteligente a vizinhança, diminuindo o tempo computacional. O conceito de intervalos para problemas de seqüenciamento em processadores paralelos foi apresentado, inicialmente, por (FRANÇA *et al*., 1994). Além do uso de intervalos, alia-se o conceito de eficiência para melhor guiar a busca através do espaço de soluções, aumentando a possibilidade de encontrar soluções de melhor qualidade. Os resultados computacionais e sua análise são apresentados na seção 5, e na seção 6 as conclusões.

ALGORITMO *MUTAT* PARA O $R||C_{MAX}$

O Algoritmo *Mutat* possui quatro fases, uma fase construtiva e três fases de melhoramento. A primeira fase realiza a alocação inicial das tarefas às máquinas. As fases 2 e 3 utilizam as bem conhecidas vizinhanças de realocação e troca. A fase 4 utiliza seqüências de três realocações, uma tarefa sai da máquina mais carregada e é realocada em uma das outras máquinas que, por sua vez, tem uma de suas tarefas realocada em outra máquina.

Para definição das tarefas passíveis de serem envolvidas em um movimento, foi adotada uma política de restrição de vizinhança. Esta política considera a divisão das tarefas em intervalos de acordo com o tempo de processamento. Deste modo é possível avaliar as tarefas com maior probabilidade de produzir uma melhora no valor de função objetivo. Dentre as tarefas candidatas, são escolhidas aquelas que apresentarem maior eficiência.

Política de intervalos

A política de intervalos consiste em agrupar as tarefas em intervalos segundo seus tempos de processamento, considerando ainda a máquina onde a tarefa está alocada e para qual máquina ela será realocada.

Na vizinhança de realocação, para reduzir C_M é necessário que uma tarefa satisfaça a desigualdade (5). Isto significa que qualquer tarefa que seja retirada da máquina M reduz C_M, mas quando esta tarefa é alocada na máquina H, o tempo de processamento desta tarefa deve ser menor que $(C_M - C_H)$, de maneira que o tempo total de processamento da máquina H não ultrapasse C_M.

A vizinhança de troca requer um pouco mais de esforço para identificar movimentos de melhoramento. A desigualdade (6) garante que a tarefa que sai da máquina M tenha um tempo de processamento maior que o tempo de processamento da tarefa que sai de máquina H, quando executada na máquina M. Desta forma C_M é reduzido. Agora, deve-se garantir que a alteração em C_H não supera C_M, e isto é feito pela desigualdade (7).

$$p_{H\,j_M} < (C_M - C_H) \quad (5)$$

$$p_{M\,j_M} - p_{M\,j_H} > 0 \quad (6)$$

$$p_{H\,j_M} - p_{H\,j_H} < d \quad (7)$$

Uma vez que se conhece o limitante superior e inferior dos tempos de processamento das tarefas envolvidas nos movimentos, uma estrutura de intervalos considerando $\underline{p_i} = \min_{j \in J}\{p_{ij}\}$ e $\overline{p_i} = \max_{j \in J}\{p_{ij}\}$, $i = 1, \ldots, m$ pode ser construída. Cada intervalo $\left[\underline{p_i}, \overline{p_i}\right]$ é dividido em aproximadamente r intervalos iguais, onde r é um parâmetro do algoritmo. Considere $V = \{v_{ik}\}$, $i = 1,\ldots, m$, $k = 1,\ldots, r$, o

conjunto dos intervalos. Cada tarefa é associada para m intervalos, um para cada máquina, de acordo com o tempo de processamento p_{ij}.

Na vizinhança de realocação, a estrutura de intervalos é usada da maneira explicada a seguir. Considere $v_{H k_0}$, $k_0 \in R$, tal que $(C_M - C_H) \in v_{H k_0}$. Para encontrar uma tarefa que satisfaça (5) deve-se examinar os intervalos v_{Hk}, $k = k_0, k_0 - 1, \ldots, 1$.

Para se utilizar a estrutura de intervalos na vizinhança de troca, adaptações mais específicas devem ser feitas. É importante observar que no $R||C_{max}$, minimizar o custo da máquina mais carregada envolvida no movimento não é equivalente a minimizar a diferença entre as duas máquinas do movimento, como ocorre no $P||C_{max}$. A estratégia que analisa os movimentos de troca adotados é a seguinte. Para cada tarefa alocada na máquina M deve-se procurar uma tarefa na máquina H que pertença ao intervalo v_{Hk}, $k = k_0, k_0 + 1, \ldots, r$, onde $\left(p_{H j_M} - d\right) \in v_{H k_0}$, e também satisfaça a desigualdade (7). Com o uso de estruturas de dados apropriadas, este procedimento produz uma significativa redução no esforço computacional, no entanto, não tão grande quanto no $P||C_{max}$.

Conceito de eficiência

Seja $m_j = \min_{1 \le i \le m} p_{ij}$ o menor tempo de processamento da tarefa j. A eficiência da máquina i para a tarefa j pode ser definida conforme a equação (8).

$$ef(i,j) = \frac{m_j}{p_{ij}} \qquad (8)$$

A eficiência é um valor de referência que indica, através de um número pertencente ao intervalo (0 , 1], qual processador executaria a tarefa mais rapidamente. Por exemplo, uma tarefa será executada em menor tempo no processador sua eficiência é igual a 1.

Observe que a restrição (2) do modelo matemático apresentado pode ser substituída por (9)

$$\sum_{j=1}^{n} \left(\frac{m_j}{ef(i,j)} x_{ij} \right) \le c, \quad \forall i \in I \qquad (9)$$

Uma vez que os valores de m_j são constantes, busca-se alocar cada tarefa à máquina com a maior eficiência possível. O Algoritmo *Mutat* será descrito a seguir.

Algoritmo *Mutat*

Fase 1: Alocam-se as tarefas, em qualquer seqüência, à máquina na qual elas possuírem eficiência igual 1. Em caso de empate, é escolhida a máquina menos carregada. Se ainda persistir o empate, escolhe-se a máquina que possuir o menor índice.

Esta alocação inicial pode produzir soluções altamente desbalanceadas, ou seja, uma grande diferença do tempo de processamento da máquina mais carregada e das demais.

Fase 2: Nesta fase procura-se retirar uma tarefa da máquina mais carregada e realocá-la, preferencialmente, na máquina menos carregada. Caso não seja possível um movimento envolvendo as máquinas mais e menos carregadas, é feita uma tentativa de realocação entre a máquina mais carregada e a segunda menos carregada, até que todas as m-1 máquinas menos carregadas sejam analisadas. Dentre todas as tarefas que podem minimizar a função objetivo é escolhida aquela que apresentar a maior eficiência na máquina para a qual será transferida. O procedimento é descrito a seguir:

Passo 1: Perfile as máquinas em ordem não-decrescente segundo as suas respectivas cargas. Represente a máquina mais carregada por M, e faça $h = 1$. Vá para o **Passo 2**.
Passo 2: Calcule $d = C_M - C_h$. Identifique o intervalo k_0, tal que $d \in v_{hk_0}$. Se $d < \underline{p}$, não existe movimento possível que minimize o *makespan*, então vá para o **Passo 4**. Se $d > \overline{p}$, faça $l = r$, caso contrário identifique k_0, tal que $d \in v_{hk_0}$, e faça $l = k_0$ e vá para o **Passo 3**.
Passo 3: Dentre as tarefas alocadas na máquina M, pertencentes aos intervalos $v_{hl}, v_{h\,l-1}, \ldots, v_{h\,1}$, que podem minimizar o *makespan*, identifique a tarefa com tempo de processamento $p_{M j_0}$, que possua a maior eficiência $ef(h,j)$ e faça sua transferência da máquina M para a máquina h. Se nenhuma troca foi identificada vá para o **Passo 4**, caso contrário vá para o **Passo 1**.
Passo 4: Faça $h := h + 1$, se $h < M$ vá para o Passo 2, caso contrário vá para a **Fase 3**.

Fase 3: Nesta fase procura-se uma tarefa, na máquina mais carregada M, para ser trocada com uma tarefa de uma outra máquina h . O procedimento é descrito a seguir:

Passo 1: Perfile as máquinas em ordem não-decrescente segundo as suas respectivas cargas. Represente a máquina mais carregada por M, e faça $h = 1$ e $t = M$. Vá para o **Passo 2**.
Passo 2: Calcule $d = C_t - C_h$. Para cada tarefa j_t alocada na máquina t faça:
Identifique k_0, tal que $\left(p_{h j_t} - d\right) \in v_{hk_0}$. Agora, devem-se procurar tarefas na máquina t pertencentes aos intervalos v_{hk}, $k = k_0, k_0 + 1, \ldots, r$ (essas tarefas possuem maior probabilidade de melhorar a solução).
Se existem tarefas tais que $p_{h j_h} < \left(p_{h j_t} - d\right)$ e

$p_{t j_t} - p_{t j_h} > 0$, entre as identificadas anteriormente, escolhem-se as tarefas com maior valor de soma do valor de eficiência.

Se nenhum movimento foi identificado, vá para o **Passo 3**. Caso contrário, execute o movimento de troca e retorne à **Fase 2**.

Passo 3: Faça $h := h + 1$. Se $h < t$ vá para o **Passo 2**, caso contrário, faça $t := t - 1$ e $h = 1$. Se $t > 2$ vá para o **Passo 2**, caso contrário vá para a **Fase 4.**

Fase 4: Aqui procura-se por uma seqüência de três realocações que minimize o *makespan,* isto é, uma tarefa sai da máquina mais carregada e é realocada em uma das outras máquinas que, por sua vez, tem uma de suas tarefas realocada em outra máquina. Os procedimentos desta fase são muito semelhantes aos da fase 2. Os procedimentos desta fase são descritos a seguir:

Passo 1: Perfile as máquinas em ordem não-decrescente segundo a suas respectivas cargas. Represente a máquina mais carregada por M, e faça $h = 1$. Vá para o **Passo 2.**

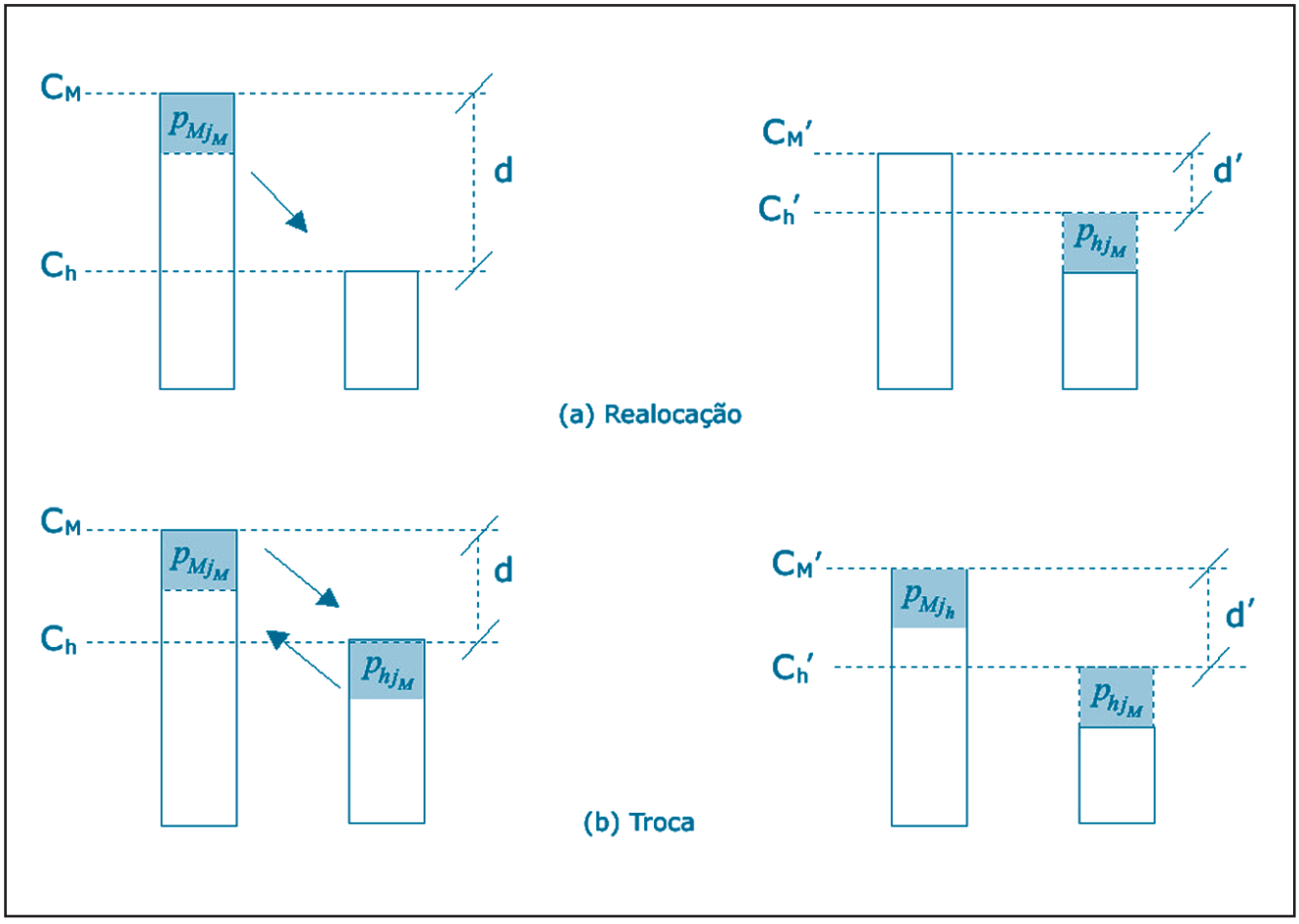
Passo 2: Se todas as tarefas da máquina M já foram consideradas, vá para o **Passo 6.** Caso contrário, transfira uma tarefa da máquina M para a máquina h. A máquina h passa a ser a nova máquina *makespan*, vá para o **Passo 3.**

Passo 3: Perfile as máquinas em ordem não-decrescente segundo a suas respectivas cargas. Represente a máquina mais carregada por M', e faça $t = 1$. Vá para o **Passo 4.**

Passo 4: Calcule $d = C_{M'} - C_h$. Identifique o intervalo k_0, tal que $d \in v_{hk_0}$. Se $d < \underline{p}$, não existe movimento possível que minimize o *makespan*, então vá para o **Passo 6**. Se $d > \overline{p}$, faça $l = r$, caso contrário identifique k_0, tal que $d \in v_{hk_0}$, e faça $l = k_0$ e vá para o **Passo 3**.

Passo 5: Dentre as tarefas pertencentes aos intervalos $v_l, v_{l-1}, \ldots, v_1$, que podem minimizar o *makespan* (C_M), identifique a tarefa com tempo de processamento $p_{M' j_0}$, que possui a maior eficiência $ef(t, j)$ e faça sua transferên-

Figura 1: Representação gráfica de um movimento de realocação (a) e troca (b).



cia da máquina *M'* para a máquina *h.* Se nenhuma realocação foi identificada vá para o **Passo 6**, caso contrário vá para o **Passo 3.**

Passo 6: Faça $t = t + 1$, se $t < M'$ vá para o **Passo 4,** caso contrário vá para a **Passo 7.**

Passo 7: Se uma realocação foi identificada no **Passo 5,** vá para o **Passo 1**, caso contrário faça $h = h + 1$. Se $h < M$ vá para o **Passo 2**, caso contrário PARE.

A figura 1(a) exemplifica um movimento de realocação, onde uma tarefa da máquina mais carregada, *M*, é realocada em outra máquina, *h*, obtendo-se com isto uma carga $C_{M'} < C_M$. Obviamente, poder-se-ia obter, dependendo das tarefas envolvidas na realocação, $C_{h'} < C_M$. Da mesma forma, a figura 1(b) exemplifica o movimento de troca entre duas máquinas. Já a figura 2 mostra uma seqüência de realocações, onde se evidencia o fato de os movimentos utilizados na Fase 4 serem compostos por uma seqüência de realocações.

RESULTADOS COMPUTACIONAIS

O Algoritmo *Mutat* foi comparado com o algoritmo heurístico proposto por (PIERSMA; DIJK, 1996), cujo desempenho foi o melhor encontrado na literatura para o problema $R||C_{max}$. Este algoritmo será chamado de Algoritmo de Eficiência. Os testes foram feitos usando o conjunto de dados proposto por (GLASS *et al.*, 1994). As instâncias foram geradas considerando as seguintes três estruturas para os tempos de processamento:

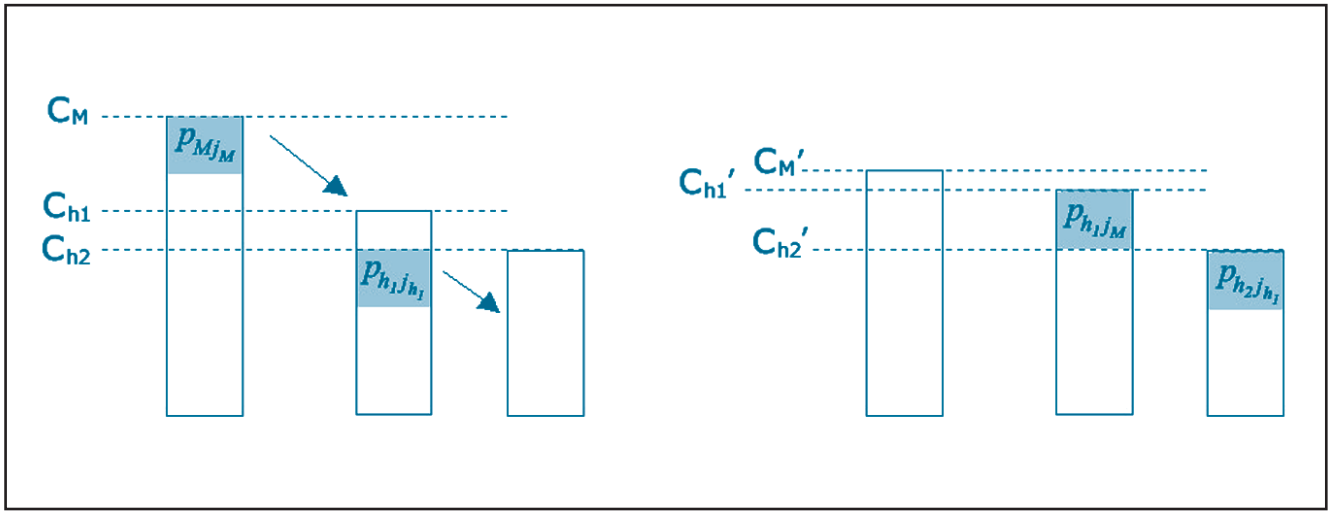
1. **TP1** – Não existe correlação entre processadores ou tarefas, os tempos de processamento p_{ij} são gerados obedecendo a uma distribuição uniforme discreta no intervalo [1 ; 100];
2. **TP2** – As tarefas são correlacionadas, os tempos de processamento p_{ij} são gerados obedecendo uma distribuição uniforme discreta no intervalo $[\beta_j+1; \beta_j+20]$, sendo β_j gerado obedecendo a uma distribuição uniforme discreta no intervalo [1 ; 100];
3. **TP3** – Os processadores são correlacionados, os tempos de processamento p_{ij} são gerados obedecendo a uma distribuição uniforme discreta no intervalo $[\alpha_i+1 ; \alpha_i+20]$, sendo α_i gerado obedecendo a uma distribuição uniforme discreta no intervalo [1 ; 100].

Para cada uma das estruturas propostas foram geradas dez instâncias de cada combinação de número de tarefas, n = 50, 100, 150 e 200, e de número de processadores, m = 2, 3, 5, 10, 25 e 50. Os resultados apresentados são uma média dos dez resultados obtidos para cada combinação. A melhor solução conhecida para o problema foi obtida utilizando as meta-heurísticas descritas em (GLASS *et al.*, 1994) e (PIERSMA; VAN DIJK, 1996), com o tempo limite de 500 segundos como critério de parada.

O Algoritmo *Mutat* foi implementado em linguagem C padrão e o Algoritmo de Eficiência em Turbo Pascal 7.0. Todos os testes foram realizados em um microcomputador K6II 550 MHz, com 126 Mb de memória RAM.

Os valores recomendados para r são apresentados na Tabela 1. A seguinte legenda é utilizada para a construção das tabelas 2 e 3:

Figura 2: Representação gráfica de uma seqüência de realocações.



AM (%)	Desvio médio para o Algoritmo *Mutat*.
EF (%)	Desvio médio para o Algoritmo de Eficiência.
TAM (s)	Tempo computacional do Algoritmo *Mutat* em segundos.
TEF (s)	Tempo computacional do Algoritmo de Eficiência em segundos.

O desvio médio em relação à melhor solução conhecida foi calculado segundo (11).

$$100 * \frac{C_{max} - C^*_{max}}{C^*_{max}} \qquad (11)$$

Onde, C_{max} representa a solução encontrada pelo Algoritmo *Mutat* ou pelo Algoritmo de Eficiência e C^*_{max} a melhor solução conhecida.

A Tabela 2 apresenta os resultados do desvio médio em relação à melhor solução conhecida, considerando todos os conjuntos de dados e a Tabela 3, tempo de CPU em segundos.

Os dois algoritmos, *Mutat* e de Eficiência, utilizam o mesmo procedimento para construção da solução inicial. O Algoritmo de Eficiência utiliza as vizinhanças de realocação e troca, fazendo a busca na vizinhança considerando primeiro as tarefas com maior eficiência. Isto faz com que este procedimento necessite de várias ordenações, aumentando sua complexidade. Já o Algoritmo *Mutat* utiliza estrutura de intervalos, evitando a ordenação de tarefas e restringindo o tamanho das vizinhanças, conseqüentemente, tem-se uma considerável redução no esforço computacional. Outra diferença entre os algoritmos é a sinergia entre as vizinhanças utilizada no Algoritmo *Mutat*, pois ele só pára quando se esgotam as possibilidades de troca nas Fases 2 e 3, caso contrário, a cada troca realizada na Fase 3 ele retorna ao início da Fase 2.

Observa-se que o Algoritmo *Mutat* supera o Algoritmo de Eficiência, na média da qualidade de solução obtida, em todas as estruturas de instâncias geradas. Porém, dos 72 casos apresentados, que totalizam 720 problemas, em 4 casos os resultados foram idênticos e em 56 combinações específicas apresentou resultados superiores. A Tabela 2 mostra em negrito os casos onde os algoritmos apresentam o melhor resultado. Com relação ao tempo computacional, o Algoritmo *Mutat* é aproximadamente cinco vezes mais rápido, na média, que o Algoritmo de Eficiência, fazendo com que, se houver necessidade de uma maior qualidade na solução, uma metaheurística possa ser aplicada. A maior diferença no desempenho dos algoritmos ocorre para o conjunto de dados do tipo TP3. Ainda, o tempo computacional do Algoritmo de Eficiência para o conjunto de dados do tipo TP3 é bem superior ao encontrado, por este mesmo algoritmo, para os conjuntos de dados TP1 e TP2. Isto ocorre porque as instâncias pertencentes ao conjunto de dados TP3 exigem um maior número de movimentos da vizinhança de trocas, sendo esta a maior vizinhança. Neste caso em específico, o custo de ordenação de tarefas fica mais evidente.

CONCLUSÕES

Neste trabalho foi apresentado um algoritmo heurístico, denominado Algoritmo *Mutat*, para a resolução do problema $R||C_{max}$. O Algoritmo *Mutat* apresentou um tempo de CPU significativamente menor que o Algoritmo de Eficiência e uma melhor qualidade das soluções para todas as estruturas de instâncias geradas. O ganho de tempo computacional do Algoritmo *Mutat* deve-se à estrutura de intervalos que este utiliza para pesquisar as vizinhanças, diferente do Algoritmo de Eficiência, que ordena as tarefas, segundo

Tabela 1: Valores recomendados para *r*.

n/m	r
(1,10)	2
(10,50)	7
(50,100)	12
(100,200)	16
(200, ...	20

Tabela 2: Desvio médio das soluções em relação à melhor solução conhecida.

		TP1		TP2		TP3	
m	*n*	**AF** (%)	**AE** (%)	**AF** (%)	**AE** (%)	**AF** (%)	**AE** (%)
2	50	**0,127**	**0,127**	0,121	**0,064**	0,072	**0,045**
	100	**0,127**	0,152	0,038	**0,034**	**0,036**	0,059
	150	**0,056**	0,087	**0,016**	0,027	0,022	**0,019**
	200	**0,030**	**0,030**	0,014	**0,009**	**0,009**	0,017
3	50	0,829	**0,711**	0,148	**0,138**	**0,077**	0,128
	100	**0,205**	0,217	0,076	**0,043**	**0,090**	0,110
	150	**0,103**	**0,103**	0,058	**0,022**	**0,090**	0,094
	200	**0,124**	0,154	**0,016**	0,022	**0,034**	0,147
5	50	**1,116**	1,172	0,248	**0,177**	**0,308**	0,531
	100	0,386	**0,359**	**0,009**	0,044	**0,641**	1,106
	150	**0,152**	0,208	**0,037**	0,049	**0,357**	0,672
	200	**0,100**	0,115	**0,000**	0,005	**0,465**	0,556
10	50	**3,010**	3,679	0,725	**0,689**	**0,772**	2,150
	100	**0,383**	0,575	**0,000**	0,148	**0,869**	1,358
	150	**0,000**	0,066	**0,025**	0,074	**1,340**	2,194
	200	**0,000**	0,050	**0,000**	0,029	**0,163**	1,035
25	50	**2,073**	7,772	**3,663**	4,174	**0,657**	2,365
	100	**0,000**	2,381	**0,564**	0,799	**0,987**	2,398
	150	**0,000**	1,538	**0,155**	0,404	**0,000**	1,373
	200	**0,000**	0,744	**0,070**	0,117	**0,000**	1,961
50	50	**2,198**	5,495	**0,000**	0,000	**0,000**	1,969
	100	**5,357**	7,143	**1,298**	2,855	**0,000**	1,395
	150	**2,308**	**2,308**	**0,871**	2,427	**0,000**	0,793
	200	**0,000**	3,676	**0,278**	0,833	**0,000**	1,006
Média Total		**0,778**	1,619	**0,351**	0,549	**0,291**	0,978

Tabela 3: Tempo de CPU em segundos.

		TP1		TP2		TP3	
m	*n*	**TAF** (s)	**TAE** (s)	**TAF** (s)	**TAE** (s)	**TAF** (s)	**TAE** (s)
2	50	0,007	0,110	0,008	0,000	0,008	0,000
	100	0,013	0,600	0,017	1,100	0,047	0,000
	150	0,027	0,600	0,043	0,500	0,061	0,600
	200	0,077	0,600	0,064	1,100	0,084	1,100
3	50	0,016	0,000	0,017	0,600	0,026	0,500
	100	0,073	0,000	0,089	0,600	0,104	1,100
	150	0,115	0,000	0,121	0,600	0,283	3,900
	200	0,210	1,000	0,274	1,000	0,414	2,970
5	50	0,022	0,600	0,025	1,100	0,047	0,000
	100	0,066	1,100	0,106	1,700	0,261	3,910
	150	0,169	1,600	0,251	1,000	0,682	2,610
	200	0,343	2,200	0,392	2,300	1,223	3,080
10	50	0,048	0,000	0,046	0,500	0,142	1,600
	100	0,130	0,000	0,218	1,100	0,678	1,430
	150	0,385	2,300	0,380	1,000	2,230	5,160
	200	0,519	2,700	0,665	2,100	4,301	16,020
25	50	0,108	0,600	0,178	0,500	0,469	4,230
	100	0,333	1,100	0,448	4,900	1,826	5,610
	150	0,970	5,000	0,695	3,840	4,398	16,200
	200	1,477	4,070	1,268	2,030	9,386	46,770
50	50	0,333	0,600	0,88	4,810	0,914	3,180
	100	0,546	3,300	1,483	2,140	3,371	10,950
	150	0,605	4,800	2,087	2,500	9,157	50,780
	200	2,139	3,840	2,568	3,690	18,289	111,330
Média Total		0,351	1,530	0,513	1,696	2,433	12,210

o critério de eficiência, a cada iteração. A melhora na qualidade de solução deve-se, principalmente, à sinergia entre as vizinhanças de troca e realocação.

A partir desse estudo e de outras propostas para esse problema, verifica-se a necessidade de se explorarem ainda mais as vizinhanças, remetendo ao desenvolvimento de metaheurísticas para o problema $R||C_{max}$. Um estudo preliminar sobre aplicação de metaheurísticas para o problema $R||C_{max}$ pode ser encontrado em (MÜLLER; ARAÚJO, 2001).

Referências bibliográficas

BAKER, K.R. *Introduction to Sequencing and Scheduling*. New York: John Wiley & Sons Inc., 1974.

CHO, Y.; SHANI, S. Bounds for list schedules on uniforms processors scheduling. *SIAM Journal on Computing*, Vol. 9, pp. 91-103, 1980.

COFFMAN Jr., E.G.; GAREY, M.R.; JOHNSON, D.S. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, Vol. 7, pp. 1-17, 1978.

DOBSON, G. Scheduling independent tasks on uniforms processors. *SIAM Journal on Computing*, Vol. 13, pp. 705-716, 1984.

FINN, G.; HOROWITZ, E. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, Vol. 19, pp. 312-320, 1979.

FRANÇA, P.M.; GENDREAU M.; LAPORTE G.; MÜLLER F.M. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective, *Computers Ops*. Res. 21, pp.205-210, 1994.

FRIESEN, D.K.; LANGSTON, M.A. Bounds for MULTIFIT scheduling on uniform processors. *SIAM Journal on Computing*, Vol. 12, pp. 60-70, 1983.

FRIESEN, D.K.; LANGSTON, M.A. Evaluation of a MULTIFIT-based scheduling algorithm. *Journal of Algorithms*, Vol. 7, pp. 35-59, 1986.

GAREY, M.R.; JOHNSON, D.S. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. San Francisco : W. H. Freeman and Company, 1979.

GLASS, C.A.; POTTS, C.N.; SHADE, P. Unrelated parallel machine scheduling using local search, *Mathl. Comput. Modelling* 20 (2), pp 41-52.54, 1994.

GONZALEZ, T.; IBARRA, O.H.; SAHNI, S. Bounds for LPT scheduling on uniforms processors. *SIAM Journal on Computing*, Vol. 6, pp. 155-166, 1977.

GRAHAM, R.L. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, Vol. 17, pp. 416-429, 1969.

HOROWITZ, E.; SANHI, S.K. Exact and approximation algorithms for scheduling nonidentical processors. *J. Assoc. Comput.*, Vol. 23, p. 317-327, 1976.

LANGSTON, M.A. Improved 0/1 interchange scheduling. *BIT*, Vol. 22, pp. 282-290, 1982.

LAWLER, E.L.; LENSTRA, J. K.; RINNOOY KAN, A.H.G.; SHMOYS, D.B. *Sequencing and scheduling: algorithms and complexity*, Report BS-R8909, Center for Mathematics and Computer Science, Amsterdan, 1989.

LIMBERGER, S.J. *Algoritmos heurísticos e exatos para solução de problemas de seqüenciamento em processadores paralelos uniformes*. Tese de Mestrado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Maria, UFSM, 1997.

LIU, J. W. S.; LIU, C.L. Bounds on scheduling algorithms for heterogeneous computing system. *Information Processing* 74, pp. 349-353, 1974.

MARTELLO, S.; SOUMIS, F.; TOTH, P. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics* 75, pp. 169-188, 1997.

McNAUGHTON, R. Scheduling with deadlines and loss functions. *Management Science*, Vol. 6(1), pp. 1-12, 1959.

MÜLLER, F.M. *Algoritmos heurísticos e exatos para resolução do problema de seqüenciamento em processadores paralelos*. Tese de Doutorado, Faculdade de Engenharia elétrica, Universidade Estadual de Campinas. Campinas, 1993.

MÜLLER, F.M.; ARAÚJO, O.C.B. Neighbourhood Constraint for a Tabu Search Algorithm to Schedule Jobs on Unrelated Parallel Machines. *4th Metaheuristics International Conference*, Porto-Portugal, pp. 551-555, 2001.

MÜLLER, F.M.; CAMOZZATO, M.M.; ARAÚJO, O. C. B. Exact Algorithms for the Imbalanced Time Minimizing Assignment Problem. *Brazilian Symposium on Graphs, Algorithms and Combinatorics*, Fortaleza-Brasil, pp. 172-175, 2001.

PIERSMA, N.; VAN DIJK, W. A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search. *Mathl. Comput. Modelling* 24 (9), pp. 11-19, 1996.

Agradecimentos

Agradecemos a valiosa colaboração dos avaliadores do trabalho, que permitiram uma melhora considerável no texto dele. Ao Prof. Dalton Varela Tubino, pela revisão do texto. O trabalho do Prof. Felipe Martins Müller foi parcialmente financiado pelo CNPq.