UMA HEURÍSTICA DE TROCAS PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM PROCESSADORES UNIFORMES

Felipe Martins Müller
Universidade Federal de Santa Maria
Centro de Tecnologia
Departamento de Eletrônica e Computação
97105-900 Santa Maria - RS
Fone:(055)220-8523 FAX:(055)220-8030
e-mail: felipe@inf.ufsm.br

Sergio João Limberger
Universidade Federal de Santa Maria
Centro de Processamento de Dados
97105-900 Santa Maria - RS
Fone:(055)220-8603 FAX:(055)226-1481
e-mail: sergio@cpd.ufsm.br

Resumo

O sequenciamento de tarefas independentes de forma não preemptiva em sistemas de processadores uniformes, com o objetivo de minimizar o tempo total de execução (*makespan*), é o assunto do presente artigo. Considera-se um conjunto de *n* tarefas, onde cada tarefa possui um tempo de processamento, e um conjunto $m \geq 2$ de processadores com velocidades de processamento $\sigma_1 = 1 \leq \sigma_2 \leq ... \leq \sigma_m$. Sendo o problema de encontrar o mínimo *makespan* considerado NP-difícil, desenvolveu-se uma heurística de trocas poderosa para resolvê-lo. A heurística proposta é composta de três fases: alocação inicial, balanceamento de carga e fase de dupla troca. A principal característica desta nova heurística é a de prescindir de uma pré-ordenação das tarefas. A heurística desenvolvida foi comparada com um limitante inferior da solução ótima e também com outras três heurísticas apresentando um desempenho superior, encontrando a solução ótima em um grande número de casos as custas de um baixo esforço computacional.

Palavras-chave: Problemas de Seqüenciamento, Otimização Combinatória, Heurísticas.

Abstract

This paper examines the nonpreemptive assignment of independent jobs to a system of uniform processors with the objective of reducing makespan, or the time required from the start of execution until all jobs are completed. We consider a set of *n* jobs, each having an execution time and a set of $m \geq 2$ processors which are assumed to have different speeds (say $\sigma_1 = 1 \leq \sigma_2 \leq ... \leq \sigma_m$). Since the problem of finding a minimal makespan has been show to be NP-hard, we develop a powerful interchange heuristic. The heuristic proposed is composed by three phases: initial assignment, job reassignment and job interchange. The main feature of this method is not perform a pre-classification of the tasks. Some comparison are made with other heuristic schemes and a lower bound that validates the results obtained. The heuristic achieve optimal solutions for several instances in a short computational time.

Keywords: scheduling problems, combinatorial optimization, heuristics.

1 INTRODUÇÃO

Neste artigo propõe-se um novo procedimento heurístico para o problema de sequenciamento de *n* tarefas independentes $J_1, J_2, \ldots, J_n$ em *m* processadores uniformes $M_1, M_2, \ldots, M_m$, sem preempção. Cada tarefa J_j tem um tempo de processamento positivo p_j e cada processador tem uma velocidade positiva σ_i que são normalizadas de modo que $\sigma_1 = 1 \leq \sigma_2 \leq \sigma_3 \leq \ldots \leq \sigma_m$.

Sendo C_i o tempo de finalização do processador M_i, significando o somatório dos tempos de processamento de todas as tarefas alocadas ao processador M_i dividido pela velocidade deste processador, denominada de σ_i. Então, o problema é determinar a alocação das tarefas que minimize o tempo máximo de finalização do processador mais carregado (*makespan*), definido como, $C_{max} = \max_{i=1,\ldots,m} \{ C_i \}$. Usando a classificação de três campos introduzida por Graham et al. (1979), o problema é definido como sendo o $Q \mid \mid C_{max}$. Considerando o caso em que as velocidades $\sigma_1 = \sigma_2 = \ldots = \sigma_m = 1$, tem-se o problema de sequenciamento em processadores paralelos idênticos.

Vários exemplos podem ser encontrados no mundo real, tais como: indústrias com um parque de máquinas de diversas gerações e, no ambiente computacional, o problema pode ser considerado quando se trata de sistemas de processamento distribuído e em redes heterogêneas. Muitos outros exemplos de problemas reais podem ser encontrados, modelados e resolvidos como um $Q \mid \mid C_{max}$.

O propósito do presente artigo é o de apresentar a heurística KPROC, um procedimento composto por três fases: uma fase construtiva e duas fases de melhoramento. Este algoritmo é descrito na seção 2. Os resultados computacionais são apresentados na seção 3. Os resultados computacionais indicam que o algoritmo proposto apresenta resultados muito próximos a um limitante inferior, encontrando a solução ótima em um grande número de casos, superando os resultados obtidos por outras heurísticas.

A maioria dos algoritmos desenvolvidos para o problema $Q \mid \mid C_{max}$ são adaptações de heurísticas inicialmente desenvolvidas para solucionar o problema $P \mid \mid C_{max}$. Estas heurísticas podem, de maneira genérica, ser classificadas em heurísticas construtivas e heurísticas de melhoramento. Grande parte destes algoritmos são do tipo construtivo, conforme pode ser observado nos estudos sobre heurísticas exatas e aproximadas apresentados por Horowitz & Sahni (1976) e Lawler et al. (1989).

Uma das heurísticas construtivas mais conhecida para o $P \mid \mid C_{max}$ é a desenvolvida por Graham (1969), denominada LPT *(Longest Processing Time first*). LPT propõe um sequenciamento através da alocação das tarefas aos processadores obedecendo uma ordem não crescente dos seus tempos de processamento. Considera-se inicialmente a alocação da tarefa de maior tempo de processamento e então aloca-se cada uma das tarefas restantes da seqüência ao processador que primeiro ficar ocioso até que todas elas estejam alocadas.

Duas adaptações da heurística LPT foram propostas para o problema $Q \mid \mid C_{max}$, baseadas nos estudos iniciados por Liu & Liu (1974). Na primeira variante, apresentada por Gonzalez et al. (1977) e Dobson (1984) em trabalhos independentes, ordenam-se as tarefas em ordem não crescente do seu tempo de processamento. Considera-se inicialmente a alocação da tarefa de maior tempo de processamento e então, seguindo a lista ordenada, a próxima tarefa considerada para a alocação é designada ao processador que primeiro ficar ocioso. Em igualdade de condições a tarefa é alocada ao processador de maior velocidade de processamento.

A segunda variante do algoritmo LPT, apresentada por Morrison (1988), mantém a ordenação das tarefas, mas as tarefas são alocadas ao processador que primeiro ficar ocioso, uma vez processada a tarefa a ser alocada. Em igualdade de condições em dois ou mais processadores, a tarefa é alocada ao processador de maior velocidade.

Outro algoritmo originalmente proposto para o problema $P||C_{max}$ e também adaptado para o problema $Q||C_{max}$ é o MULTIFIT, Coffman et al. (1978). A adaptação é proposta por Friesen & Langston (1983) e Kunde (1983), também em trabalhos independentes, esta implementação transforma o problema de sequenciamento de tarefas em um problema de empacotamento (*bin packing problem*). Para transformar o problema de sequenciamento em um problema de empacotamento, considera-se que cada processador corresponde a uma caixa, no caso de processadores idênticos, a caixa mais carregada corresponde ao processador mais carregado. As tarefas a serem processadas correspondem aos itens a serem armazenados. Estendendo esta analogia ao caso de processadores uniformes, define-se o tamanho das caixas como variável, sendo função da velocidade do processador. O tamanho da caixa *i* é definida como sendo o valor determinado para a capacidade de cada caixa multiplicado por σ_i , que corresponde a velocidade do processador *i.*

O algoritmo de empacotamento FFD (*First Fit Decreasing*) faz uma pré-ordenação dos itens a serem alocados em ordem não crescente de seu tamanho. No momento da alocação, o item é designado à primeira caixa, na qual este é passível de alocação (não excede a capacidade da mesma) ou, então, uma nova caixa é criada. Na implementação do algoritmo MULTIFIT necessita-se de um limitante superior e um limitante inferior para o tamanho das caixas utilizadas no algoritmo FFD. Então faz-se um processo iterativo atualizando os limitantes inferiores e superiores da solução, considerando as seguintes situações: no caso de falta de espaço para a alocação das tarefas, necessita-se aumentar o tamanho das caixas e caso haja sobra de espaço, diminui-se o tamanho das caixas.

2 A HEURÍSTICA KPROC

O algoritmo heurístico proposto KPROC, pode ser resumido como: na fase 1, as tarefas são classificadas de acordo com seu tempo de processamento e designadas aos processadores de forma a obter uma distribuição razoavelmente balanceada. Na fase 2, um balanceamento de carga é efetuado pela movimentação de uma tarefa do processador mais carregado para o processador menos carregado. Finalmente na fase 3, um balanço ainda melhor é efetuado através da troca de tarefas a serem efetuadas, entre o processador mais carregado e um dos outros processadores. Uma descrição mais detalhada das fases do algoritmo KPROC é realizada a seguir.

Fase 1: Alocação inicial. Diferente de diversos algoritmos construtivos conhecidos, não é necessário uma ordenação das tarefas em função do seu tempo de processamento. A vantagem de não se fazer a ordenação inicial das tarefas é que se elimina a maior parcela da complexidade computacional deste tipo de algoritmo, podendo utilizar o tempo economizado para realizar outras tarefas. Além disso a divisão das tarefas em intervalos facilita bastante a busca das tarefas a serem movidas ou trocadas, eliminando muitas pesquisas desnecessárias.

Para realizar a divisão das tarefas em subintervalos, deve-se conhecer apenas os limitantes inferior e superior dos valores dos tempos de processamento, ou seja, os valores de $\underline{p} = \min\{p_j\}$ e $\overline{p} = \max\{p_j\}$. Então, $[\underline{p}, \overline{p}]$ é dividido em *k* intervalos iguais $I_1, I_2, \ldots, I_k$, onde *k* é um parâmetro de entrada do algoritmo. A tarefa J_j é chamada de uma *l*-tarefa se $p_j \in I_l$.

Inicialmente inicializa-se $n_{ir} = 0$, $i = 1, 2, \ldots, m$ e $r = 1, 2, \ldots, k$, onde n_{ir} é o número de tarefas do intervalo *r* designadas ao processador *i*. Então aloca-se cada *l*-tarefa ao processador com a menor razão de n_{il}/σ_i. Em caso de igualdade, a tarefa é alocada ao processador que tiver maior velocidade. Esta política busca um balanceamento de carga por intervalo, ou seja, o número de tarefas alocadas a cada intervalo é proporcional a velocidade do mesmo.

Fase 2: Fase de Balanceamento. Seja

$$\overline{C} = \sum_{i=1}^{n} pi \Big/ \sum_{i=1}^{m} \sigma i$$

o tempo médio de finalização, independente em relação a uma distribuição particular das tarefas aos processadores. Esta fase consiste na repetição do movimento de transferência de tarefas do processador mais carregado (com o maior C_i), para o processador menos carregado, tendo como alvo $\overline{C}$, buscando reduzir o tempo de finalização do processador mais carregado (*makespan*). O procedimento é descrito a seguir.

Passo 1 - Identifique os processadores mais e menos carregados e chame-os de M_1 e M_m, respectivamente.

Passo 2 - Calcule $\Delta_1 = C_1 - \overline{C}$, $\Delta_m = \overline{C} - C_m$, $\Delta = \min\{\Delta_1,\Delta_m\}$ e $\Delta' = C_1 - C_m$. O cálculo de Δ é utilizado para que se possa identificar quais tarefas poderão trazer melhorias.

Se $\sigma_m \Delta < \underline{p}$, Termina a FASE 2. Isto significa que não existem tarefas que, se executadas no processador *m*, produziriam um novo C_m menor ou igual a Δ. Observe que a velocidade do processador *m*, σ_m , é utilizada para que possamos ter a noção exata do impacto que o tempo de processamento de uma tarefa teria no tempo de finalização deste processador (por exemplo, se a tarefa J_j com tempo de processamento p_j fosse processada em M_m, produziria um acréscimo de p_j / σ_m no tempo de finalização).

Se $\sigma_m \Delta > \overline{p}$ faça l := *k* (o maior intervalo). Isto significa que qualquer uma das tarefas, se executada no processador *m*, produz um novo C_m menor ou igual a Δ.

Caso contrário $\sigma_m \Delta$ pertence a algum intervalo I_l. Ou seja, identifico o intervalo a partir do qual devo buscar tarefas que são possíveis de produzir um novo C_m menor ou igual a Δ.

Passo 3 - Se nenhuma tarefa J_j com $p_j \in I_l$ e $p_j \leq \sigma_m \Delta$ está alocada em M_l, vá para o passo 4. Caso contrário realoque a tarefa com $p_j \in I_l$ de M_1 em M_m e vá para o passo 1. Neste passo busca-se uma tarefa no intervalo de maior índice para ser trocada, neste intervalo tem-se as tarefas que produzem novos C_m mais próximos de Δ.

Passo 4 - Se nenhuma tarefa J_j com $p_j \in I_l = I_1 \cup I_2 \cup ... \cup I_{l-1}$ está alocada em M_1, vá para o passo 5. Caso contrário, realoque a tarefa com $p_j \in I_l$ de M_l em M_m e vá para o passo 1. Aqui busca-se uma tarefa nos intervalos de índice menor que *l*, pois é o local onde ainda posso obter uma tarefa que produza um novo C_m menor ou igual a Δ quando alocadas a M_m.

Passo 5 - **Se $l = k$ vá para a FASE 3.** Não há tarefas possíveis de ser alocadas em M_m produzindo um novo C_m menor ou igual a Δ.

Se nenhuma tarefa J_j com $p_j \in I_l^+ = \{ \cup I_i, i = l + 1, ... / p_i < \sigma_m \Delta'$ ou $i = k \}$ e $p_j < \sigma_m \Delta'$, vá para a FASE 3, caso contrário, realoque uma tarefa de I_l^+ de M_l para M_m e vá para o passo 1. Aqui busca-se nos intervalos de índice maior que *l* uma tarefa que produza um novo C_m menor que Δ', ou seja, não atinge o alvo, mas ainda assim diminui o *makespan*.

Observa-se que as tarefas são agrupadas em conjuntos de acordo com o seu tempo de processamento, de tal forma a tornar mais eficiente a busca das tarefas candidatas a transferência entre os processadores

Fase 3: Fase de duplas trocas. Novamente M_1 e M_m referem-se aos dois processadores, mais e menos carregados respectivamente. (Os demais processadores são aleatoriamente relacionados). Nesta fase procura-se uma tarefa para troca do processador mais carregado M_1 (a tarefa chamada J_j) com uma tarefa de um outro processador M_h (chamada de tarefa J_j'), começando pelo processador $h = m, m-1, ...$, até que uma tarefa seja encontrada.

A idéia é delimitar o intervalo de tarefas nos processadores alvo que possibilitem uma redução no valor de solução para cada tarefa do processador mais carregado, para isto busca-se um valor que minimize a diferença de tempos de processamento entre M_1 e M_h.

O valor de θ representa a diferença em tempo entre as de tarefas J_j e J_j' que possibilitam aos processadores envolvidos atingirem o equilíbrio de suas cargas $\overline{C}$ (que é o alvo da busca nesta fase do algoritmo), ou seja, $\theta = p_j - p_{j'}$. Ele é determinado, conforme equilíbrio de carga no final da transferência que ocorre entre os processadores envolvidos durante a fase 3.

Sendo C_1' e C_h' o valor dos tempos de finalização de M_1 e M_h , respectivamente, após a troca ser realizada, formulados como:

$$C_1' = C_1 - (p_j - p_{j'})/\sigma_1 \text{ e } C_h' = C_h + (p_j - p_{j'})/\sigma_h .$$

Portanto no ponto de equilíbrio a igualdade $C_1' = C_h'$ é válida, desenvolvendo-se esta equação e isolando o valor de θ, obtém-se $\theta = \frac{\sigma_1\sigma_h(C_1 - C_h)}{\sigma_1 + \sigma_h}$. Observa-se que o ponto de equilíbrio é função das velocidades e dos tempos de finalização de M_1 e M_h.

Portanto as tarefas disponíveis para transferência, são as que se encontram dentro dos limites obtidos pela seguinte fórmula:

$(p_j - p_{j'})\,\sigma_h < C_1 - C_h$, que, rearranjada, pode ser expressa como, $p_{j'} > p_j - \frac{C_1 - C_h}{\sigma_h}$.

Então procura-se um par de tarefas J_j e J_j' com $p_{j'} < p_j$, onde $|p_j - p_{j'}|$ é o mais próximo ao valor de θ de tal forma a melhorar a qualidade da solução.

Procurando manter uma boa relação entre o esforço computacional e a qualidade da solução o procedimento é baseado na quantidade de tarefas do problema, como descrito a seguir.

Passo 1 - Se $n < 100$ vá para o Passo 2.

Caso contrário

Identificar o primeiro par J_j' da tarefa J_j , considerando $h = m, m - 1, ...$, e de tal forma que $p_{j'} < p_j$ e que $p_{j'} > p_j - \frac{C_1 - C_h}{\sigma_h}$. Caso um par de tarefas que satisfaça estas condições for encontrado, volte a Fase 2. Caso contrário o algoritmo TERMINA.

Passo 2 - Para cada processador h = m, m-1, ... , calcular $\theta = \frac{\sigma_1\sigma_h(C_1 - C_h)}{\sigma_1 + \sigma_h}$ e identifique o par de tarefas J_j e J_j' onde $|p_j - p_{j'}|$ é mais próximo ao valor de θ (pois, quanto mais próximo do valor de θ for esta diferença, melhor será a qualidade da solução) de tal forma que $p_{j'} < p_j$ e que $p_{j'} > p_j - \frac{C_1 - C_h}{\sigma_h}$. Caso um par de tarefas que satisfaça a condição for encontrado, volte a Fase 2. Caso contrário o algoritmo TERMINA.

Considerando o fato que a heurística pode interagir uma quantidade indeterminada de vezes entre as fases 2 e 3, não é possível obter facilmente a avaliação da complexidade de pior caso do algoritmo. Pode-se empiricamente observar a boa performance do algoritmo KPROC em relação aos demais algoritmos.

Os resultados obtidos são apresentados na seção 3 a seguir.

3 RESULTADOS COMPUTACIONAIS

A heurística KPROC foi comparada com três outras heurísticas conhecidas:

(1) LPTU1, uma variante do LPT proposta por Gonzalez et al. (1977) e Dobson (1984).

(2) LPTU2, uma variante do LPT proposta por Morrison (1988).

(3) MFIT, uma variante do MULTIFIT proposto por Friesen & Langston (1983) e Kunde (1983).

Usou-se o termo variante porque todas as três implementações de heurísticas são adaptações originalmente propostas para o solução do problema $P||C_{max}$ como afirmado anteriormente.

As quatro heurísticas são comparadas umas com as outras, e estas são comparadas com um limitante inferior definido como $\overline{C}$ (limitante inferior do valor ótimo). Também forma feitas algumas comparações com a solução ótima, porém com um conjunto diferente de problemas teste, pois o procedimento ótimo só aceita valores inteiros.

Testes foram executados para o problema com m = 2, 3, 7, 10, 15, 20 e n = 10, 50, 100, 500, 1000 e tempos de processamento nos intervalos [1,100], [1,1000] e [1,10000], com distribuição uniforme, considerando valores inteiros e valores reais. A escolha destes problemas se deve ao fato de se buscar um algoritmo que seja bastante robusto e não tenha restrição quanto aos valores dos tempos de processamento. Os valores do tipo real são uma constante neste tipo de problema, uma vez que divisões entre o tempo de processamento e a velocidade do processador no qual ele vai ser processados são uma necessidade para se determinar a solução do problema. Os resultados aqui apresentados referem-se a distribuição uniforme e valores inteiros nos intervalos [1,100] e [1,10000] (tabelas 2 e 3), pois estes representam bem o desempenho do algoritmo.

Primeiro foi analisado o efeito do parâmetro k, o número de intervalos dos tempos de processamento, no desempenho do algoritmo. Foram testados diversos valores para k, dentro do intervalo [1,28]. Dentre estes, observando-se as melhores relações entre qualidade da solução e esforço computacional para obtê-la, definiu-se o k em função do valor da relação (m/n). Os valores recomendados de k^* são apresentados na Tabela 1, considerando o valor de solução e o tempo computacional necessário para a obtenção da melhor solução segundo apresentado abaixo.

n*/*m	***k****
[0, 10)	3
[10,50)	7
[50,100)	12
[100,200)	16
[200,300)	20

[300,400)	24
[400,500)	28

Tabela 1 - Valores Recomendados de k

Em todos os problemas apresentados foi usado $k = k^*$ e todos os casos foram testados para um conjunto de 10 instâncias teste.

As variáveis utilizadas nas Tabelas 2, 3 e 4 são definidas abaixo:

LB : média dos valores de

$$\overline{C} = \sum_{i=1}^{n} pi \Big/ \sum_{i=1}^{m} \sigma i ,$$

considerando dez instâncias teste.

KPROC, LPTU1, LPTU2 e MFIT : média dos valores de solução encontrados por KPROC, LPTU1, LPTU2 e MFIT, respectivamente, considerando dez instâncias teste.

SO : média dos valores de solução ótima, considerando dez instâncias teste.

$$\overline{\mathbf{KPROC}} \;:\; \left(\mathbf{KPROC} - \mathbf{LB} \big/ \mathbf{LB}\right) \times 100$$

$$\overline{\mathbf{LPTU1}} \;:\; \left(\mathbf{LPTU1} - \mathbf{LB} \big/ \mathbf{LB}\right) \times 100$$

$$\overline{\mathbf{LPTU2}} \;:\; \left(\mathbf{LPTU2} - \mathbf{LB} \big/ \mathbf{LB}\right) \times 100$$

$$\overline{\mathbf{MFIT}} \;:\; \left(\mathbf{MFIT} - \mathbf{LB} \big/ \mathbf{LB}\right) \times 100$$

TKPROC, TLPTU1, TLPTU2 e TMFIT : média dos tempos de CPU, em segundos, gastos por KPROC, LPTU1, LPTU2 e MFIT, respectivamente, considerando dez instâncias teste.

m	n	$\overline{\text{KPROC}}$	$\overline{\text{LPTU1}}$	$\overline{\text{LPTU2}}$	$\overline{\text{MFIT}}$	TKPROC	TLPTU1	TLPTU2	TMFIT
2	10	0,462	0,856	1,749	0,856	0,003	0,002	0,001	0,001
	50	0,021	0,079	0,088	0,059	0,002	0,002	0,001	0,023
	100	0,008	0,017	0,022	0,011	0,005	0,003	0,005	0,085
	500	0,002	0,002	0,003	0,002	0,043	0,065	0,060	1,812
	1000	0,001	0,002	0,002	0,002	0,095	0,237	0,228	6,810
3	10	1,849	2,761	5,177	3,337	0,005	0,001	0,001	0,003
	50	0,041	0,181	0,391	0,076	0,005	0,005	0,002	0,022
	100	0,021	0,047	0,078	0,032	0,008	0,003	0,007	0,070
	500	0,005	0,004	0,008	0,005	0,040	0,050	0,048	1,232
	1000	0,001	0,001	0,002	0,001	0,083	0,165	0,167	5,052
5	50	0,095	0,586	0,733	0,248	0,005	0,005	0,002	0,018
	100	0,037	0,116	0,159	0,046	0,018	0,008	0,005	0,062
	500	0,007	0,008	0,010	0,007	0,036	0,040	0,033	0,790
	1000	0,003	0,003	0,005	0,004	0,112	0,122	0,105	3,470
7	50	0,158	0,885	1,345	0,392	0,017	0,003	0,001	0,022
	100	0,064	0,192	0,332	0,082	0,021	0,002	0,003	0,053
	500	0,011	0,011	0,016	0,012	0,033	0,033	0,027	0,767
	1000	0,006	0,006	0,007	0,006	0,090	0,097	0,088	2,332
10	50	0,418	1,938	2,621	0,754	0,020	0,005	0,002	0,022
	100	0,110	0,367	0,616	0,137	0,031	0,008	0,003	0,058
	500	0,017	0,025	0,034	0,018	0,045	0,032	0,020	0,608
	1000	0,009	0,009	0,014	0,010	0,097	0,092	0,070	2,215
15	50	1,456	3,273	7,609	1,203	0,038	0,005	0,002	0,037
	100	0,184	0,848	1,718	0,281	0,060	0,012	0,010	0,073
	500	0,028	0,044	0,069	0,030	0,050	0,040	0,025	0,595
	1000	0,014	0,015	0,023	0,014	0,102	0,093	0,060	1,857
20	50	2,433	6,572	15,132	15,450	0,060	0,003	0,003	0,037
	100	0,293	1,493	2,998	0,367	0,053	0,013	0,003	0,085
	500	0,036	0,082	0,142	0,038	0,060	0,047	0,018	0,590
	1000	0,018	0,020	0,028	0,021	0,122	0,092	0,055	1,660

Tabela 2 - Resultados Computacionais para Valores Inteiros no Intervalo [1,100]

m	n	$\overline{\text{KPROC}}$	$\overline{\text{LPTU1}}$	$\overline{\text{LPTU2}}$	$\overline{\text{MFIT}}$	TKPROC	TLPTU1	TLPTU2	TMFIT
2	10	0,275	2,057	1,357	1,008	0,003	0,001	0,001	0,003
	50	0,002	0,064	0,080	0,032	0,002	0,003	0,001	0,023
	100	0,000	0,013	0,026	0,011	0,012	0,001	0,001	0,082
	500	0,000	0,001	0,001	0,000	0,035	0,067	0,067	1,798
	1000	0,000	0,000	0,000	0,000	0,078	0,240	0,242	7,275
3	10	2,029	3,617	8,148	2,011	0,002	0,002	0,001	0,001
	50	0,018	0,213	0,190	0,068	0,007	0,003	0,001	0,020
	100	0,001	0,032	0,101	0,018	0,015	0,002	0,005	0,060
	500	0,000	0,002	0,002	0,001	0,043	0,048	0,048	1,292
	1000	0,000	0,000	0,001	0,000	0,092	0,173	0,167	5,050
5	50	0,052	0,442	0,778	0,211	0,012	0,001	0,001	0,023
	100	0,008	0,142	0,153	0,028	0,035	0,007	0,002	0,053
	500	0,000	0,003	0,007	0,001	0,052	0,042	0,032	0,885
	1000	0,000	0,001	0,002	0,000	0,090	0,120	0,118	3,400
7	50	0,234	0,872	1,441	0,296	0,015	0,003	0,002	0,023
	100	0,025	0,212	0,369	0,074	0,043	0,007	0,008	0,057
	500	0,000	0,007	0,015	0,003	0,047	0,037	0,037	0,775
	1000	0,000	0,002	0,004	0,001	0,107	0,110	0,092	2,733
10	50	0,405	1,354	3,305	0,482	0,023	0,002	0,003	0,025
	100	0,033	0,331	0,855	0,117	0,065	0,007	0,002	0,057
	500	0,000	0,015	0,031	0,005	0,058	0,043	0,027	0,623
	1000	0,000	0,004	0,009	0,001	0,113	0,102	0,075	2,140
15	50	1,132	4,295	7,105	1,185	0,052	0,010	0,002	0,030
	100	0,152	1,030	2,125	0,249	0,097	0,007	0,005	0,073
	500	0,001	0,033	0,060	0,010	0,080	0,045	0,022	0,652
	1000	0,000	0,007	0,019	0,002	0,140	0,093	0,063	1,883
20	50	2,932	5,753	13,059	9,827	0,062	0,005	0,001	0,043
	100	0,246	1,425	4,129	0,372	0,143	0,010	0,003	0,087
	500	0,002	0,059	0,124	0,011	0,100	0,047	0,023	0,628
	1000	0,000	0,013	0,032	0,003	0,176	0,098	0,060	1,758

Tabela 3 - Resultados Computacionais para Valores Inteiros no Intervalo [1,10000]

As quatro heurísticas foram implementadas em linguagem C, testadas e executadas em uma estação de trabalho IBM RISC/6000 Modelo 25T.

Os resultados indicam que a heurística KPROC produz valores de solução, geralmente inferiores a 1% do valor do limitante inferior, que significa que a solução é muito próxima aos valores ótimos da solução. O algoritmo KPROC, comparado com as três outras heurísticas, apresenta na absoluta maioria das instâncias apresentadas o melhor resultado.

A heurística KPROC é também mais rápida que a MFIT, sendo que apresenta, na relação de tempo de CPU, a mesma magnitude apresentada pelas heurísticas LPTU1 e LPTU2, conhecidas por seu bom desempenho computacional. Em resumo, a heurística proposta produz soluções ótimas ou quase ótimas, com tempo computacional relativamente baixo e apresenta a melhor solução entre as heurísticas comparativas apresentadas. Para confirmar ainda mais esta afirmação, desenvolveu-se um esquema ótimo para solução deste problema baseado na analogia com o problema de alocação generalizada (PAG).

O PAG pode ser descrito, usando-se a terminologia do problema da mochila, ou seja, dados n itens e m mochilas, com p_{ij} sendo o lucro do item j se alocado a mochila i; w_{ij} sendo o peso do item j se alocado a mochila i; e c_i sendo a capacidade da mochila i; aloque cada item a

exatamente uma mochila de modo a maximizar o lucro total, respeitando as restrições de capacidade de cada mochila.

Este problema já apresenta uma relação com os problemas de sequenciamento, pois muitas vezes aparece utilizado para alocação ótima de n tarefas a m processadores, considerando p_{ij} o lucro; w_{ij} a quantidade de recurso correspondente a alocação da tarefa j ao processador i; e c_i a quantidade de recurso disponível em cada processador i.

No caso aqui apresentado, processadores uniformes, a analogia foi feita do seguinte modo:

- considerou-se todos os lucros $p_{ij} = 1$;
- os pesos w_{ij} foram considerados iguais a p_i / σ_j (onde p_i é o tempo de processamento da tarefa i e σ_j é a velocidade do processador j);
- a capacidade da mochila c_i é inicializada com o valor da solução encontrada pela heurística KPROC, para todo $i = 1 , ... , m$, ou seja, um limitante superior para a solução ótima.

Para se encontrar a solução ótima do problema, utilizou-se um código para resolução do PAG denominado MTG, desenvolvido por Martello & Toth (1990). Considerou-se a analogia apresentada anteriormente e seguiu-se o seguinte procedimento iterativo para obter a solução ótima.

Passo 1 - Faça a Solução Ótima receber, inicialmente, a Solução encontrada pela heurística KPROC.

Faça o valor de c_i ser igual ao valor da solução encontrada pela heurística KPROC subtraída de uma unidade, para todo $i = 1 , ... , m$.

Passo 2 - Resolva o PAG correspondente.

Passo 3 - Se a solução do PAG for factível, então

- Faça a Solução Ótima ser igual a Solução encontrada pelo PAG;
- Faça o valor de c_i ser igual ao valor da solução encontrada pelo PAG subtraída de uma unidade, para todo $i = 1 , ... , m$; e volte para o passo 2.

Se a solução do PAG for infactível, então, apresente a solução ótima e PARE.

Uma vez que o código MTG só trabalha com valores inteiros, foram gerados alguns novos conjuntos de dados para realizar os testes comparativos da heurística KPROC com a solução ótima. Para cada combinação de $m = 2 , 3$ e $n = 10 , 20 , 50 , 100$, dez instâncias são geradas, considerando $\sigma_i = i,$ para $i = 1 , ... , m$ e p_j, para $j = 1 , ... , n$, gerado aleatoriamente no intervalo [1,100] considerando uma distribuição uniforme. Os tempos de processamento gerados devem obedecer a restrição de serem divisíveis por $\sigma_i = i,$ para $i = 1 , ... , m$, para que toda alocação, independente da velocidade do processador, gere tempos de finalização inteiros.

Uma síntese destes resultados é apresentada na tabela 4, onde os resultados apresentados são a média de dez instâncias teste. Desta tabela, pode-se notar que a maioria das soluções encontradas por KPROC é ótima. De fato, nas 80 instâncias testadas, foi observado somente um caso ($m = 2$ e $n = 10$) em que o resultado encontrado por KPROC foi uma unidade superior ao valor da solução ótima.

m	n	KPROC	SO	$\left(\text{KPROC} - \text{SO}/\text{SO}\right)\times 100$
2	10	161,8	161,7	0,062
	20	335,3	335,3	0,000
	50	878,7	878,7	0,000
	100	1753,0	1753,0	0,000
3	10	88,4	88,4	0,000
	20	176,6	176,6	0,000
	50	420,8	420,8	0,000
	100	834,0	834,0	0,000

Tabela 4 - Resultados Comparativos entre a Solução Ótima e a Heurística KPROC

4 CONCLUSÃO

A heurística KPROC demonstrou ser um algoritmo robusto em comparação com os outros métodos testados e com o limitante inferior. Também demonstrou um bom desempenho obtendo soluções bastante próximas da ótima e, em muitos casos a própria solução ótima, as custas de um esforço computacional razoável.

Agradecimentos – Agradecemos aos dois revisores anônimos pelos seus comentários e sugestões que tanto valorizaram este trabalho. Agradecemos também a acadêmica do Curso de Informática da UFSM, Tatiane Jesus de Campos, pelo seu empenho em executar e tabular os dados relativos à solução ótima. O trabalho do Prof. Felipe Martins Müller foi parcialmente financiado pelo CNPq e pela FAPERGS.

5 REFERÊNCIAS BIBLIOGRÁFICAS

(1) Cho, Y. & Sahni, S. (1980). Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, **9**, 91-103.

(2) Coffman Jr., E.G., Garey, M.R. & Johnson, D.S. (1978). An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, **7**, 1-17.

(3) Dobson, G. (1984). Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, **13**, 705-716.

(4) França, P.M., Gendreau, M., Laporte, G. & Müller, F.M. (1994). A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers & Operations Research*, **21**, 205-210.

(5) Friensen, D.K. & Langston, M.A. (1983). Bounds for MULTIFIT scheduling on uniform processors. *SIAM Journal on Computing*, **12**, 60-70.

(6) Friesen, D.K. & Langston, M.A. (1986). Evaluation of a MULTIFIT-based scheduling algorithm. *Journal of Algorithms*. **7**, 35-59.

(7) Garey, M.R. & Johnson, D.S. (1979). *Computers Intractability: A Guide to the Theory of NP- Completeness*. Freeman, San Francisco.

(8) Gonzalez, T., Ibarra, O.H. & Sahni, S. (1977). Bounds for LPT schedules on uniform processors. *SIAM Journal on Computing*, **6**, 155-166.

(9) Graham, R.L. , Lawler, E.L., Lenstra, J.K. & Rinnooy Kan, A.H.G. (1979). Optimization and approximation in determinist sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, **5**, 287-326.

(10) Hochbaum, D.S. & Shmoys, D.B. (1988). A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, **17**, 539-551.

(11) Horowitz, E. & Sahni, S. (1976). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the Association for Computer Machinery*, **23**, 317-327.

(12) Kunde, M. (1983). A MULTIFIT algorithm for uniform multiprocessor scheduling. *Lecture Notes in Computer Science*, **145**, 175-185.

(13) Kunde, M., Langston, M.A and Liu, J. (1988). On a special case of uniform processor scheduling. *Journal of Algorithms*, **9**, 287-296.

(14) Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. & Shmoys, D.B. (1989). Sequencing and scheduling: algorithms and complexity. *Report BS-R8909*, *Center of Mathematics and Computer Science*, Amsterdan.

(15) Liu, J.W.S. & Liu, C.L. (1974). Bounds on scheduling algorithms for heterogeneous computing system. *Information Processing*, **74**, 349-353.

(16) Martello, S. & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons Ltd., Chichester, England.

(17) Morrison, J.F. (1988). A note on LPT scheduling, *Operations Research Letters*, **7**, 77-79.