TEMA Tend. Mat. Apl. Comput., **4**, No. 1 (2003), 89-98.

Inovações na Solução do Problema de Seqüenciamento em Processadores Paralelos Uniformes

F.M. MÜLLER, H.G. SANTOS, V.C. KÖHLER, A. MARIN[1]; Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Produção e Curso de Ciência da Computação, 97105-900 Santa Maria, RS, Brasil.

Resumo.

O problema abordado neste artigo trata da alocação de um conjunto finito de tarefas em um determinado número de máquinas que possuem diferentes velocidades de processamento, de forma não preemptiva, com o objetivo de minimizar o tempo total de execução das tarefas (*makespan*). A abordagem de solução escolhida nesse trabalho foi o método heurístico, conhecido na literatura como *Kproc* [7], no qual foram propostas modificações que resultaram em um ganho na qualidade de solução em 99,98% dos tipos de instâncias considerados nos experimentos, sem um comprometimento significativo do tempo computacional. O método modificado foi denominado *Iproc*. Os dois métodos heurísticos mencionados foram comparados com um limitante inferior e foram implementados através do *framework* para problemas de otimização combinatória CORE (*Combinatorial Optimization Resource Management Environment*), o qual possibilita a execução remota e distribuída através da estrutura da *Internet* de métodos de resolução para diversos problemas de otimização combinatória. O ambiente CORE pode ser acessado em http://glover.ce.ufsm.br.

1. Introdução

Considerando o seqüenciamento de n tarefas independentes, de forma não preemptiva, com tempos de processamento p_j, $j \in J = \{1, ..., n\}$ a m máquinas paralelas uniformes $M_1, M_2, ..., M_m$ com velocidades de processamento $\sigma_1 = 1 \leq \sigma_2 \leq \sigma_3 \leq ... \leq \sigma_m$, e sendo T_i a soma dos tempos de processamento das tarefas alocadas no processador M_i e C_i o tempo total de execução do processador M_i, obtido através de T_i/σ_i, o objetivo do problema é minimizar o tempo necessário para processar todas as tarefas (*makespan*) definido como $C_{max} = max\ \{C_i\}$, para $i = 1, ..., m$. Este problema é denominado, segundo a classificação de três campos proposta por [6], $Q||C_{max}$.

[1]felipe,haroldo,viviane,marin@inf.ufsm.br,

O problema tem várias aplicações práticas tanto no ambiente industrial quanto no científico. Um exemplo pode ser encontrado num parque de máquinas onde a atualização ocorre sem a dispensa do maquinário legado. Outro exemplo pode ser o escalonamento de processos em *clusters* com recursos computacionais heterogêneos.

Considerando o caso em que $\sigma_1 = \sigma_2 = ... = \sigma_m = 1$, tem-se o problema de seqüenciamento de máquinas paralelas idênticas ou $P||C_{max}$.

Grande parte dos métodos heurísticos utilizados para a resolução do problema $Q||C_{max}$ são adaptações de heurísticas inicialmente desenvolvidas para o problema $P||C_{max}$. As heurísticas utilizadas no presente artigo podem, de forma genérica, ser classificadas em heurísticas construtivas e heurísticas de melhoramento. Enquanto as primeiras montam, passo a passo, uma solução, as heurísticas de melhoramento tentam, através de movimentos sistemáticos, melhorar a qualidade de solução recebida.

Entre as heurísticas construtivas mais conhecidas para o $P||C_{max}$, podemos citar a desenvolvida por [4], denominada LPT (*Longest Processing Time first*). A partir dessa, [3] e [1], em trabalhos independentes, propuseram uma adaptação do LPT para o $Q||C_{max}$, outra variante do LPT foi proposta por [8].

Seguindo essa idéia, [7] desenvolveram a heurística *Kproc*, a qual consiste em uma adaptação do algoritmo *3-Fases* proposto por [9] para resolução do $P||C_{max}$.

2. A heurística *Kproc*

A heurística *Kproc* baseia-se no conceito de intervalos, ou seja, as tarefas a serem alocadas são separadas de acordo com seu tamanho e o número de intervalos é definido utilizando uma função discreta que recebe como parâmetro a razão entre a quantidade de tarefas e o número de processadores disponíveis. A separação das tarefas em intervalos substitui a ordenação inicial, procedimento existente na maioria dos algoritmos tradicionais.

A estrutura intervalar utilizada pelo *Kproc* assegura uma alocação equilibrada de carga em cada intervalo de cada processador. Além disso, restringe o espectro de possíveis trocas vantajosas das tarefas entre os processadores.

A execução do método *Kproc* ocorre em três fases distintas: alocação inicial, balanceamento de carga e fase de dupla troca. Na primeira, considerada a fase construtiva do algoritmo, é efetuada a distribuição das tarefas aos processadores. Na segunda fase, parte-se da solução encontrada pela alocação inicial, e procura-se a redução do tempo de finalização do processador mais carregado, transferindo tarefas deste para o processador menos carregado. Na última fase, permite-se a permutação de tarefas a partir de um alvo definido *a priori*, entre o processador mais carregado e os outros processadores.

Tabela 1: Função discreta para a definição do número de intervalos

n/m	< 10	[10,50)	[50,100)	[100,200)	[200,300)	[300,400)	[400,500)
k	3	7	12	16	20	24	28

Fases do *Kproc*

Fase 1

Passo 1 – Calcular o limitante inferior $\underline{p} = \min_j\{p_j\}$ e o limitante superior $\overline{p} = \max_j\{p_j\}$ dos tempos de processamento das tarefas sendo $j = 1, 2, ..., n$;

Passo 2 – Criar k intervalos, com tamanhos aproximadamente iguais, variando entre $[\underline{p}, \overline{p}]$, o valor de k é definido segundo a tabela 1. Para cada processador, associar um conjunto I de intervalos, representados por $I_1, I_2, ..., I_k$;

Passo 3 – Enquanto existirem tarefas a serem alocadas, faça: Determine o intervalo I_l que contém a tarefa j. Aloque j no intervalo I_l do processador que apresentar a menor relação entre o número de tarefas alocadas no intervalo ($n(I_l)$), pela velocidade deste processador (σ_i). Em caso de igualdade na relação, a tarefa é alocada ao processador de maior velocidade.

Fase 2

Passo1 – Identifique os processadores mais e menos carregados e chame-os de M_1 e M_m, respectivamente.

Passo 2 – Considerando: C_1 a carga do processador mais carregado; C_m a carga do processador menos carregado; e $\overline{C} = \sum_{j=1}^{n} p_j \Big/ \sum_{i=1}^{m} \sigma_i$ o tempo médio de finalização, utilizado como limitante inferior nos resultados apresentados ao final[5]. Calcule $\Delta_1 = C_1 - \overline{C}$, $\Delta_m = \overline{C} - C_m$, $\Delta = min\{\Delta_1 - \Delta_m\}$ e $\Delta' = C_1 - C_m$.

Se $\sigma_m\Delta < \underline{p}$, não existem tarefas que produzem um *makespan* melhor, portanto a Fase 2 termina.

Se $\sigma_m\Delta > \overline{p}$ faça $l = k$ (o maior intervalo).

Caso contrário, $\sigma_m\Delta$ pertence a algum intervalo I_l.

Passo 3 – Se nenhuma tarefa j com $p_j \in I_l$ e $p_j \leq \sigma_m\Delta$ está alocada em M_1, vá para o Passo 4. Caso contrário mova a primeira tarefa encontrada, dita l, de M_1 para o intervalo I_l em M_m e volte para o Passo 1.

Passo 4 – Se nenhuma tarefa j com $p_j \in I_{l^-} = I_l \cup I_2 \cup ... \cup I_{l-1}$ está alocada em M_1, vá para o passo 5. Caso contrário, mova a primeira tarefa encontrada com $p_j \in I_{l^-}$ de M_1 para o intervalo I_{l^-} em M_m e volte para o Passo 1.

Passo 5 – Se $l = k$ vá para Fase 3.

Se nenhuma tarefa j com $p_j \in I_{l^+} = \{\cup I_i, i = l, l+1, ... | p_i < \sigma_m\Delta'$ ou $i = k\}$ e $p_j < \sigma_m\Delta'$, vá para a Fase 3, caso contrário, mova uma tarefa de I_{l^+} de M_1 para M_m e vá para o Passo 1.

Fase 3

Passo 1 – 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 e j' onde $|p_j - p_j{}'|$ é mais próximo ao valor de θ de tal forma que $p_j{}' < p_j$ e que $p_j{}' > p_j - (C_1 - C_h)/\sigma_h$. Caso um par de tarefas que satisfaça a condição for encontrado, realize a troca e volte à Fase 1. Caso contrário o algoritmo termina.

3. A heurística *IProc*

Visando melhorar a qualidade de solução sem comprometer o tempo computacional, foram estudados melhoramentos para a versão original da heurística *Kproc*, principalmente no tocante aos conceitos de maior relevância presentes nesse procedimento: a distribuição das tarefas em intervalos e a estratégia utilizada para a busca local.

Após uma avaliação dos valores de k que apresentam melhor relação entre desempenho e qualidade da solução, foi proposta uma função contínua baseada nos valores empíricos discretos definidos por [7]. Como indicado abaixo:

$$k = 30 \cdot \log\ (\ 1 + 5\ (^{x}/_{1000})\ ) + 5, \tag{3.1}$$

onde $x = n/m$, ou seja, é a relação entre o número de tarefas e o de processadores.

Num segundo momento, realizou-se uma modificação na fase de balanceamento da carga (Fase 2), alterando a estratégia de busca por tarefas candidatas à troca, que originalmente selecionava a primeira tarefa cuja troca reduzisse a carga do processador mais carregado, por uma política onde faz-se uma simulação do efeito da troca aos processadores envolvidos e escolhe-se a que minimizar a carga simulada do processador mais carregado.

Na Fase 3, modificou-se o valor alvo de busca das tarefas candidatas à troca dupla do método original por um procedimento que, como o anterior, calcula o efeito da permutação das tarefas selecionadas aos processadores envolvidos e seleciona o par de tarefas que minimizar a carga estimada do processador mais carregado.

Fases do *Iproc*

Fase 1

A única alteração nessa fase refere-se à função que determina o número de intervalos, que passam a ser calculados através da equação (3.1).

Fase 2

Passo 1 – Identifique os processadores mais e menos carregados (M_1 e M_m, respectivamente).

Passo 2 – Calcule $\Delta = C_1 - C_m$ e $\Phi = \sigma_m \Delta$, onde Φ será tratado como um valor limitante para a busca

Passo 3 – Se $\Phi < \underline{p}$, a Fase 2 termina. Caso contrário, identifique o intervalo que contém o valor de Φ e chame de I_f. Procure por tarefas alocadas em M_1 com $p_j \in I_f (f = f, f-1, ..., 1)$ e $p_j \leq \Phi$. Se tais tarefas não existirem, a Fase 2 termina. Senão, selecione a tarefa que minimizar o valor da função *dife*, representada a seguir.

$$dife = \left| \overline{C} - \left(C_m - {}^{p_j}/_{\sigma_m} \right) \right|,$$

onde: p_j = tempo de processamento de uma tarefa do processador mais carregado;

σ_m = velocidade do processador menos carregado;

$\overline{C}$ = tempo médio de finalização, proposto por [5].

Realoque a tarefa j, selecionada, de M_1 para o intervalo I_f em M_m e volte para o Passo 1.

Fase 3
Passo 1 – Identifique os processadores mais e menos carregados (M_1 e M_m, respectivamente) e nomeie arbitrariamente os processadores restantes como sendo: $m-1, m-2, ..., 2$ onde m é o número de processadores.
Passo 2 – Se $m = 1$, finaliza a Fase 3. Caso contrário, para cada tarefa j, do processador M_1, procure, em M_m, uma tarefa j' que satisfaça as seguintes condições: $p_j{}' < p_j$ e $p_j{}' > p_j - (C_1 - C_m)/\sigma_m$. Caso não existir nenhum par de tarefas que satisfaça as condições, faça $m = m - 1$ e retorne ao Passo 2.
Caso contrário, para cada par de tarefas encontrado, calcule: $C_1{}' = C_1 - (p_j - p_j{}')/\sigma_1$, $C_m{}' = C_m - (p_j - p_j{}')/\sigma_m$ e $swap = \{max(C_1{}', C_m{}')\}$
Selecione o par que minimizar o valor da função *swap*, realize a permutação das tarefas selecionadas e volte para a Fase 2.

Antes de se apresentar os resultados computacionais, vai-se apresentar resumidamente o ambiente CORE, onde todos os algoritmos e instâncias aqui apresentados estão disponíveis, tanto para execução remota quanto para *download*.

4. O projeto CORE

O projeto CORE (*Combinatorial Optimization Resource Management Environment*) consiste em um ambiente acessível pela Internet que suporta a execução remota distribuída de diferentes métodos de resolução de problemas de otimização combinatória.

O ambiente oferece um conjunto de métodos heurísticos e metaheurísticos, que podem ser dinamicamente combinados para a obtenção de melhores resultados na solução de problemas de grande interesse científico e industrial.

Utilizando-se interfaces gráficas de fácil utilização, é possível a criação de **planos de otimização**, que consistem em estratégias para a solução de problemas (seleção de métodos e configuração de seus respectivos parâmetros) que serão executados sobre instâncias de teste ou instâncias enviadas pelo usuário. O ambiente foi desenvolvido através do paradigma de orientação a objetos e objetos distribuídos[2].

Mais informações sobre o projeto CORE podem ser encontradas em [10] e na página do projeto (http://glover.ce.ufsm.br).

5. Resultados Computacionais

Para avaliação do método *Iproc* proposto, utilizou-se a comparação com o método *Kproc* original. Os dois métodos foram comparados com o limitante inferior proposto por [5]. Foram geradas 10 instâncias para cada combinação de $m = 2$ e 3 e n= 10, 50, 100, 500 e 1000; e m = 5, 7, 10, 15 e 20 e n= 50, 100, 500 e 1000, com tempos de processamento sendo gerados de forma uniformemente distribuída nos intervalos [1,100], [1,1000] e [1,1000], considerando um conjunto com números inteiros e outro com números reais, totalizando 1800 instâncias. As velocidades das máquinas foram geradas obedecendo a uma distribuição uniforme no intervalo [2,20].

Como os resultados obtidos para as diversas amplitudes de tempos de processamento apresentaram resultados semelhantes, vão-se apresentar apenas os referentes aos dados com tempos de processamento no intervalo [1,100]. A tabela 2 mostra os resultados para cada combinação de m e n, onde s_1 é a média da diferença percentual entre o valor de função objetivo obtido pelo método *Kproc* e o limitante inferior, para as 10 instâncias consideradas; s_2 é a média da diferença percentual entre o valor de função objetivo obtido pelo método *Iproc* e o limitante inferior, para as 10 instâncias consideradas; t_1 é a média do tempo computacional (em segundos) para execução de *Kproc*, para as 10 instâncias consideradas e t_2 é a média do tempo computacional (em segundos) para execução de *Iproc*, para as 10 instâncias consideradas.

Os testes foram executados em um computador Pentium III 1 Gigahertz, com 384 Megabytes de *RAM*, sob o sistema operacional Linux com kernel 2.4.3.

Observou-se que, na quase totalidade dos casos, o desempenho de *Iproc* foi superior a *Kproc* as custas de um mínimo acréscimo de esforço computacional. Os casos onde isso não ocorreu se referem a (m,n) = (2,50) e (3,100) com tempos de processamento reais uniformemente distribuídos nos intervalos [1,1000] e [1,10000], respectivamente.

São apresentados também, em forma de gráficos, os desempenhos médio, máximo e o desvio padrão de *Kproc*(s_1) e *Iproc*(s_2) para m = 2, 3, 5, 7, 10, 15 e 20 e n = 50 e 100 (Figuras 1 e 2, respectivamente) com tempos de processamento inteiros uniformemente distribuídos no intervalo [1,100]. Pode-se observar que *Iproc* apresenta sempre um melhor desempenho médio, menor desvio padrão e menor resultado máximo, quando comparado ao *Kproc*, demonstrando sua robustez. Para se demonstrar as mesmas conclusões é apresentada, ainda, a Figura 3, que demonstra o desempenho médio de *Kproc*(s_1) e *Iproc*(s_2) para valores de tempos de processamento inteiros.

6. Conclusões

A qualidade de solução da heurística *Iproc* foi superior em todas as combinações, excetuando-se a relação processador/tarefa (2,50) das tarefas geradas no intervalo [1,100] e as relações (3,100) e (3,10) do intervalo de tarefas [1,10000], o que representa um ganho em qualidade de solução de 99,98% dos casos. O tempo computacional, no entanto, foi em média 20% superior ao da heurística *Kproc*, o que deve-se a exploração mais refinada das trocas possíveis na Fase 2. Portanto, pode-se dizer que o método apresentou melhores resultados, tanto médios quanto máximos às custas de um pequeno esforço computacional adicional.

A disponibilização dos métodos através do ambiente *CORE* facilitou a execução dos testes e acelerou o processo de desenvolvimento, permitindo que um grupo maior de usuários participasse do processo. Além disso, obteve-se reutilização de código e

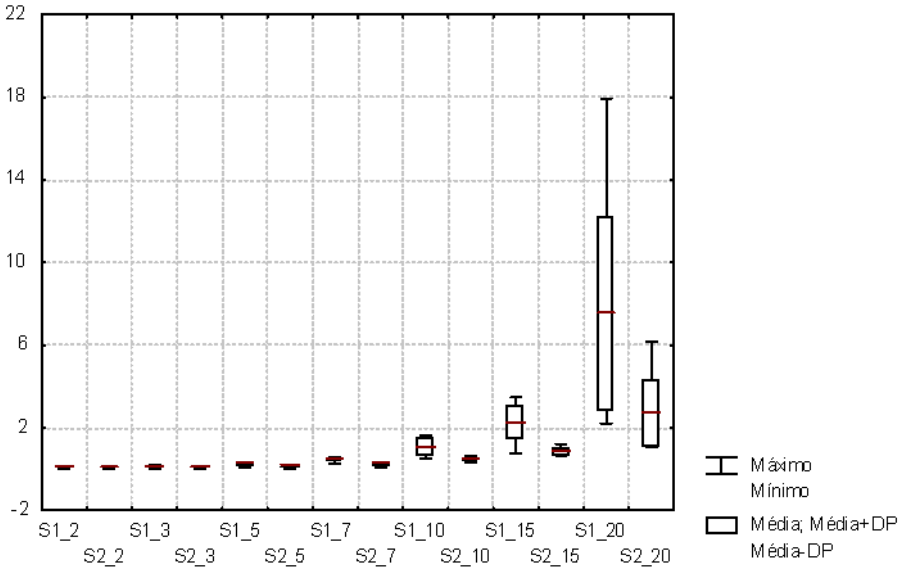

Figura 1: Desempenho de $Kproc(s_1)$ e $Iproc(s_2)$ para n = 50 com tempos de processamentos inteiros uniformemente distribuídos no intervalo [1,100]

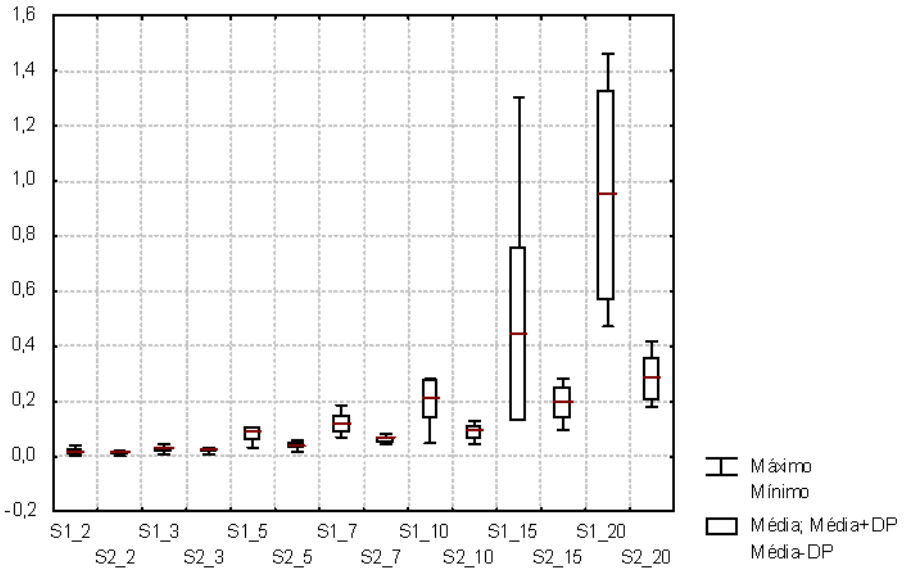

Figura 2: Desempenho de $Kproc(s_1)$ e $Iproc(s_2)$ para n = 100 com tempos de processamentos inteiros uniformemente distribuídos no intervalo [1,100]

Tabela 2: Resultados considerando tempos de processamento gerados entre [1,100]

m	n	Instâncias Inteiras				Instâncias Reais			
		s_1	s_2	t_1	t_2	s_1	s_2	t_1	t_2
2	10	0,49386	0,22125	0,4	0,5	1,92782	0,30276	0,3	0,6
	50	0,04359	0,02791	1,4	1,3	0,00850	0,00379	2,1	2,4
	100	0,01320	0,00962	2,6	6,5	0,00313	0,00032	3,4	4,0
	500	0,00253	0,00141	13,3	10,3	0,00005	0,00004	16,0	20,7
	1000	0,00265	0,00110	15,2	22,8	0,00002	0,00001	37,7	44,1
3	10	3,15542	1,65902	0,5	0,5	4,79070	1,71343	0,7	0,4
	50	0,06562	0,03705	2,3	1,8	0,04508	0,01188	2,9	3,7
	100	0,02343	0,01679	3,4	2,7	0,00539	0,00229	5,0	6,6
	500	0,00723	0,00441	14,6	14,3	0,00006	0,00006	34,9	32,3
	1000	0,00487	0,00151	29,6	31,4	0,00002	0,00001	80,0	71,4
5	50	0,18572	0,09536	2,9	3,4	0,16600	0,07821	4,1	4,3
	100	0,08097	0,03556	4,9	4,7	0,01329	0,00656	6,7	8,2
	500	0,01429	0,00840	15,8	17,3	0,00026	0,00006	42,2	42,2
	1000	0,00747	0,00433	33,0	36,1	0,00006	0,00004	81,3	81,6
7	50	0,39191	0,19554	3,4	3,4	0,28623	0,07820	5,3	5,6
	100	0,11572	0,05999	5,3	5,9	0,04635	0,01339	9,0	12,5
	500	0,02379	0,01197	15,1	22,1	0,00052	0,00016	54,0	51,3
	1000	0,01312	0,00540	35,5	42,1	0,00010	0,00006	86,6	95,5
10	50	1,04593	0,40876	6,3	3,9	0,84235	0,30853	5,3	6,2
	100	0,20711	0,08898	6,7	7,6	0,12483	0,04851	9,9	15,4
	500	0,03408	0,02058	20,1	24,9	0,00109	0,00034	48,6	71,9
	1000	0,01941	0,00886	35,6	64,3	0,00023	0,00017	96,0	100,6
15	50	2,21228	0,82429	8,9	7,2	2,04068	0,99937	8,1	9,0
	100	0,44051	0,19147	9,4	10,2	0,28191	0,10111	17,3	22,0
	500	0,04949	0,02732	23,6	30,5	0,00276	0,00097	90,1	86,9
	1000	0,02741	0,01355	42,5	73,3	0,00048	0,00018	124,7	153,4
20	50	7,46746	2,64307	8,9	11,1	6,02549	2,29970	10,3	14,0
	100	0,94493	0,27985	13,0	14,5	0,72756	0,20102	17,4	27,2
	500	0,07607	0,04012	26,4	35,9	0,00680	0,00174	113,1	145,1
	1000	0,03791	0,01869	52,9	78,5	0,00076	0,00018	195,4	169,0

projeto no desenvolvimento dos métodos heurísticos.

Agradecimentos

Os autores agradecem as sugestões feitas pelo assessor que revisou o artigo, pois foram de grande valia para uma melhor compreensão do texto. O trabalho de Felipe Martins Müller foi parcialmente financiado pelo CNPq.

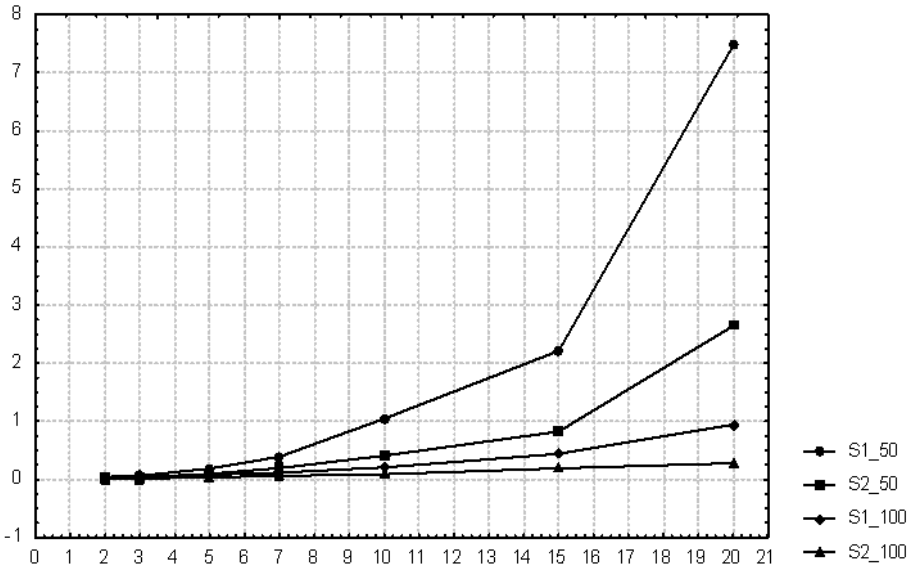

Figura 3: Gráfico comparativo do desempenho médio de $Kproc(s_1)$ e $Iproc(s_2)$ para n=50 e n=100, tempos de processamento inteiros uniformemente distribuídos no intervalo [1,100]

Abstract.

This paper deals with the problem of assigning a limited set of tasks to a given set of machines with different processing speeds, with no preemption and aiming to minimize the total execution time (makespan). The solution approach chosen in this paper was the *Kproc* heuristic, in which improvements were proposed, resulting in a solution quality gain in 99,98% of the instances types considered in the experiments, without a significant computation time increase. Still, *Kproc*, *Kproc*and many other heurístic and metaheuristc methods were implemented using the CORE (Combinatorial Optimization Resource Management Environment) framework, which allows the distributed remote execution of combinatorial optimization resolution methods through the Internet structure. The CORE environment can be accessed from: http://glover.ce.ufsm.br.

Referências

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

[2] J. Farley, "Java Distributed Computing", O'Reilly, USA, 1998.

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

[4] R.L. Graham, E.L. Lawler e J.K. Lenstra, Optimization and aproximation in determinist sequencing: a survey, *Annals of Discrete Mathematics*, **5** (1979), 287-326.

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

[6] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan e D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, Report BS-R8909, Center of Mathematics and Computer Science, 1989, Amsterdan.

[7] S. J. Limberger e F. M. Müller, Uma heurística de trocas para o problema de seqüenciamento de tarefas em processadores uniformes, *Pesquisa Operacional*, **20** (2000), 31-42.

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

[9] F.M. Müller, "Algoritmos Heurísticos e Exatos para Resolução do Problema de Seqüenciamento em Processadores Paralelos", Tese de Doutorado, DENSIS, UNICAMP, Campinas, SP, 1993.

[10] F.M. Müller e H.G. Santos, CORE - an internet based combinatorial optimization service provider em "Technical and Organizational Integration of Supply Chains (F. S. Flogliato et al., Eds.)", *Proceedings os the XXII ICIEOM*, (2002), 113-120.