A Hybrid Bin-Packing Heuristic to Multiprocessor Scheduling

Adriana C. F. Alvim[1] and Celso C. Ribeiro[2]

[1] Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Produção, PO Box 68507, Rio de Janeiro 21945-970, Brazil.
alvim@inf.puc-rio.br
[2] Universidade Federal Fluminense, Department of Computer Science, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.
celso@inf.puc-rio.br

Abstract. The multiprocessor scheduling problem consists in scheduling a set of tasks with known processing times into a set of identical processors so as to minimize their makespan, i.e., the maximum processing time over all processors. We propose a new heuristic for solving the multiprocessor scheduling problem, based on a hybrid heuristic to the bin packing problem. Computational results illustrating the effectiveness of this approach are reported and compared with those obtained by other heuristics.

1 Introduction

Let $T = \{T_1, \ldots, T_n\}$ be a set of n tasks with processing times t_i, $i = 1, \ldots, n$, to be processed by a set $P = \{P_1, \ldots, P_m\}$ of $m \geq 2$ identical processors. We assume the processing times are nonnegative integers satisfying $t_1 \geq t_2 \geq \ldots \geq t_n$. Each processor can handle at most one task at any given time and preemption is not possible. We denote by A_j the set formed by the indices of the tasks assigned to processor P_j and by $t(P_j) = \sum_{i \in A_j} t_i$ its overall processing time, $j = 1, \ldots, n$. A solution is represented by the lists of tasks assigned to each processor. The makespan of a solution $S = (A_1, \ldots, A_m)$ is given by $C_{\max}(S) = \max_{j=1,\ldots,m} t(P_j)$.

The multiprocessor scheduling problem $P||C_{\max}$ consists in finding an optimal assignment of tasks to processors, so as to minimize their makespan, see e.g. [5, 16, 20]. $P||C_{\max}$ is NP-hard [4, 14]. We denote the optimal makespan by $C^*_{\max}$. Minimizing the schedule length is important since it leads to the maximization of the processor utilization factor [3].

There is a duality relation [17, 18, 24] between $P||C_{\max}$ and the bin packing problem (BP), which consists in finding the minimum number of bins of a given capacity C which are necessary to accommodate n items with weights $t_1, \ldots, t_n$.

The worst case performance ratio $r(\mathrm{H})$ of a given heuristic H for $P||C_{\max}$ is defined as the maximum value of the ratio $\mathrm{H}(I)/z(I)$ over all instances I, where $z(I)$ is the optimal makespan of instance I and $\mathrm{H}(I)$ is the makespan of the solution computed by heuristic H. The longest processing time (LPT) heuristic of Graham [15] finds an approximate solution to $P||C_{\max}$ in time $O(n \log n + n \log m)$, with $r(\text{LPT}) = 4/3 - 1/3m$. The MULTIFIT heuristic proposed by Coffman et al. [6] explores the duality relation with the bin packing problem, searching by binary search the minimum processing time (or bin capacity) such that the solution obtained by the FFD heuristic [19, 21] to pack the n tasks (or items) makes use of at most m processors (or bins). It can be shown that if MULTIFIT is applied k times, then it runs in time $O(n \log n + kn \log m)$ and $r(\text{MULTIFIT}) = 1.22 + 2^{-k}$. Friesen [13] subsequently improved this ratio to $1.20 + 2^{-k}$. Yue [25] further improved it to 13/11, which is tight. Finn and Horowitz [9] proposed the 0/1-INTERCHANGE heuristic running in time $O(n \log m)$, with worst case performance ratio equal to 2. Variants and extensions of the above heuristics can be found in the literature.

The duality relation between the bin packing problem and $P||C_{\max}$ was also used by Alvim and Ribeiro [1] to derive a hybrid improvement heuristic to the former. This heuristic is explored in this paper in the context of $P||C_{\max}$. Lower and upper bounds used by the heuristic are described in Section 2. The main steps of the hybrid improvement heuristic to multiprocessor scheduling are presented in Section 3. Numerical results illustrating the effectiveness of the proposed algorithm are reported in Section 4. Concluding remarks are made in the last section.

2 Lower and Upper Bounds

The lower bound $L_1 = \max \left\{ \left\lceil \frac{1}{m} \sum_{i=1}^{n} p_i \right\rceil, \max_{i=1,\ldots,n}\{p_i\} \right\}$ proposed by McNaughton [22] establishes that the optimal makespan cannot be smaller than the maximum between the average processing time over all processors and the longest duration over all tasks. This bound can be further improved to $L_2 = \max \left\{ L_1,\ p_m + p_{m+1} \right\}$.

Dell'Amico and Martello [7] proposed the lower bound L_3. They also showed that the combination of lower and upper bounds to the makespan makes it possible to derive lower and upper bounds to the number of tasks assigned to each processor, leading to a new lower bound L_ϑ. Bounds L_2, L_3, and L_ϑ are used in the heuristic described in the next section.

We used three construction procedures for building feasible solutions and computing upper bounds to $P||C_{\max}$:

- Construction heuristic H1: Originally proposed in [1, 2], it is similar to the Multi-Subset heuristic in [7]. It considers one processor at a time. The longest yet unassigned task is assigned to the current processor. Next, assign to this same processor a subset of the yet unassigned tasks such that the sum of their processing times is as close as possible to a given limit to the makespan. The polynomial-time approximation scheme MTSS(3) of Martello and Toth [21] is used in this step. The remaining unassigned tasks are considered one by one in non-increasing order of their processing times. Each of them is assigned to the processor with the smallest load.

– Construction heuristic H2: Hochbaum and Shmoys [17] proposed a new approach to constructing approximation algorithms, called dual approximation algorithms. The goal is to find superoptimal, but infeasible, solutions. They showed that finding an ϵ-approximation to $P||C_{\max}$ is equivalent to finding an ϵ-dual-approximation to BP. For the latter, an ϵ-dual-approximation algorithm constructs a solution in which the number of bins is at most the optimal number of bins and each bin is filled with at most $1+\epsilon$ (bin capacity $C = 1$ and item weights t_i scaled by t_i/C). In particular, they proposed a scheme for $\epsilon = 1/5$. Given a lower bound L and an upper bound U to $C^*_{\max}$, we obtain by binary search the smallest value C such that $L \leq C \leq U$ and the 1/5-dual-approximation solution to BP uses no more than m bins. This approach characterizes a $1/5 + 2^{-k}$-approximation algorithm to $P||C_{\max}$, where k is the number of iterations of the search. At the end of the search, the value of C gives the L_{HS} lower bound to the makespan.
– Construction heuristic H3: This is the longest processing time heuristic LPT [15]. Tasks are ordered in non-increasing order of their processing times. Each task is assigned to the processor with the smallest total processing time.

França et al. [10] proposed algorithm 3-PHASE based on the idea of balancing the load between pair of processors. Hübscher and Glover [18] also explored the relation between $P||C_{\max}$ and BP, proposing a tabu search algorithm using 2-exchanges and influential diversification. Dell'Amico and Martello [7] developed a branch-and-bound algorithm to exactly solve $P||C_{\max}$. They also obtained new lower bounds. Scholl and Voss [24] considered two versions of the simple assembly line balancing problem. If the precedence constraints are not taken into account, these two versions correspond to BP and $P||C_{\max}$. Fatemi-Ghomi and Jolai-Ghazvini [8] proposed a local search algorithm using a neighborhood defined by exchanges of pairs of tasks in different processors. Frangioni et al. [12] proposed new local search algorithms for the minimum makespan processor scheduling problem, which perform multiple exchanges of jobs among machines. The latter are modelled as special disjoint cycles and paths in a suitably defined improvement graph. Several algorithms for searching the neighborhood are suggested and computational experiments are performed for the case of identical processors.

3 Hybrid Improvement Heuristic to $P||C_{\max}$

The hybrid improvement heuristic to $P||C_{\max}$ is described in this section. The core of this procedure is formed by the construction, redistribution, and improvement phases, as illustrated by the pseudo-code of procedure `C+R+I` in Figure 1. It depends on two parameters: the target makespan `Target` and the maximum number of iterations `MaxIterations` performed by the tabu search procedure used in the improvement phase. The loop in lines 1–8 makes three attempts to build a feasible solution S to the bin packing problem defined by the processing times $t_i, i = 1, \ldots, n$, with bin capacity `Target`, using exactly m bins. Each of the heuristics H1, H2, and H3 is used at each attempt in line 2. If S is feasible to the associated bin packing problem, then it is returned in line 3. Otherwise, load redistribution is performed in line 4 to improve processor usability and the modified solution S is returned in line 5 if it is feasible to the bin packing problem. Finally, a tabu search procedure is applied in line 6 as an attempt to knock down the makespan of the current solution and the modified solution S is returned in line 7 if it is feasible to the bin packing problem. Detailed descriptions of the redistribution and improvement phases are reported in [1].

```
procedure C+R+I(Target, MaxIterations);
1  for k = 1, 2, 3 do
2      Build a solution S = {A_1, ..., A_m} to P||C_max using heuristic Hk;
3      if C_max(S) ≤ Target then return S;
4      S ← Redistribution(S);
5      if C_max(S) ≤ Target then return S;
6      S ← TabuSearch(S, MaxIterations);
7      if C_max(S) ≤ Target then return S;
8  end
9  return S;
end C+R+I
```

Fig. 1. Pseudo-code of the core C+R+I procedure.

The pseudo-code of the complete hybrid improvement heuristic HI_PCmax to $P||C_{\max}$ is given in Figure 2. An initial solution S is built in line 1 using heuristic H3. The lower bound L_2 is computed in line 2. If the current lower and upper bounds coincide, then solution S is returned in line 3. The lower bound L_3 is computed in line 4 and the current lower bound is updated. If the current lower and upper bounds coincide, then solution S is returned in line 5. The lower bound L_ϑ is computed in line 6 and the current lower bound is updated. If the current lower and upper bounds coincide, then solution S is returned in line 7. A new solution S' is built in line 8 using heuristic H2. The currently best solution and the current upper bound are updated in line 9, while the current lower bound is updated in line 10. If the current lower and upper bounds coincide, then the currently best solution S is returned in line 11. A new solution S' is built in line 12 using heuristic H1. The currently best solution and the current upper bound are updated in line 13. If the current lower and upper bounds coincide, then the currently best solution S is returned in line 14. At this point, UB is the upper bound associated with the currently best known solution S to $P||C_{\max}$ and LB is an unattained makespan. The core procedure C+R+I makes an attempt to build a solution with makespan equal to the current lower bound in line 15. The currently best solution and the current upper bound are updated in line 16. If the current lower and upper bounds coincide, then the currently best solution S is returned in line 17. The loop in lines 18–23 implements a binary search strategy seeking for progressively better solutions.

The target makespan $C_{\max} = \lfloor (LB + UB)/2 \rfloor$ is set in line 19. Let S' be the solution obtained by the core procedure `C+R+I` applied in line 20 using $C_{\max}$ as the target makespan. If its makespan is at least as good as the target makespan $C_{\max}$, then the current upper bound UB and the currently best solution S are updated in line 21. Otherwise, the unattained makespan LB is updated in line 22, since the core procedure `C+R+I` was not able to find a feasible solution with the target makespan. The best solution found S is returned in line 24.

procedure `HI_PCmax(MaxIterations)`;
1 Compute a solution S using heuristic H3 and set $UB \leftarrow C_{\max}(S)$;
2 Compute the lower bound L_2 and set $LB \leftarrow L_2$;
3 **if** $LB = UB$ **then return** S;
4 Compute L_3 using binary search in the interval $[LB, UB]$ and set $LB \leftarrow L_3$;
5 **if** $LB = UB$ **then return** S;
6 Compute L_ϑ using LB and UB and update $LB \leftarrow \max\{LB, L_\vartheta\}$;
7 **if** $LB = UB$ **then return** S;
8 Compute a solution S' and the lower bound L_{HS} using heuristic H2;
9 **if** $C_{\max}(S') < UB$ **then** set $UB \leftarrow C_{\max}(S')$ and $S \leftarrow S'$;
10 Update $LB \leftarrow \max\{LB, L_{HS}\}$;
11 **if** $LB = UB$ **then return** S
12 Compute a solution S' using heuristic H1;
13 **if** $C_{\max}(S') < UB$ **then** set $UB \leftarrow C_{\max}(S')$ and $S \leftarrow S'$;
14 **if** $LB = UB$ **then return** S;
15 $S' \leftarrow$ `C+R+I`$(LB,$ `MaxIterations`$)$;
16 **if** $C_{\max}(S') < UB$ **then** set $UB \leftarrow C_{\max}(S')$ and $S \leftarrow S'$;
17 **if** $LB = UB$ **then return** S;
18 **while** $LB < UB - 1$ **do**
19 $\quad C_{\max} \leftarrow \lfloor (LB + UB)/2 \rfloor$;
20 $\quad S' \leftarrow$ `C+R+I`$(C_{\max},$ `MaxIterations`$)$;
21 $\quad$ **if** $C_{\max}(S') \leq C_{\max}$ **then** set $UB \leftarrow C_{\max}(S')$ and $S \leftarrow S'$;
22 $\quad$ **else** $LB \leftarrow C_{\max}$;
23 **end**
24 **return** S;
end `HI_PCmax`

Fig. 2. Pseudo-code of the hybrid improvement procedure to $P||C_{\max}$.

The core procedure `C+R+I` is applied at two different points: once in line 15 using the lower bound LB as the target makespan and in line 20 at each iteration of the binary search strategy using $C_{\max}$ as the target makespan. This implementation follows the same EBS (binary search with prespecified entry point) scheme suggested in [24]. Computational experiments have shown that it is able to find better solutions in smaller computation times than other variants which do not explore the binary search strategy or do not make a preliminary attempt to build a solution using LB as the target makespan.

4 Computational Experiments

All computational experiments were performed on a 2.4 GHz AMD XP machine with 512 MB of RAM memory.

Algorithms HI_PCmax and LPT were coded in `C` and compiled with version 2.95.2 of the `gcc` compiler with the optimization flag -O3. The maximum number of iterations during the tabu search improvement phase is set as `MaxIterations` = 1000. We compare the new heuristic HI_PCmax with the 3-PHASE heuristic of França et al. [10], the branch-and-bound algorithm B&B of Dell'Amico and Martello [7], and the ME multi-exchange algorithms of Frangioni et al. [12]. The code of algorithm B&B [7] was provided by Dell'Amico and Martello.

4.1 Test problems

We considered two families of test problems: uniform and non-uniform. In these families, the number of processors m takes values in $\{5, 10, 25\}$ and the number of tasks n takes values in $\{50, 100, 500, 1000\}$ (the combination $m = 5$ with $n = 10$ is also tested). The processing times $t_i, i = 1, \ldots, n$ are randomly generated in the intervals $[1, 100]$, $[1, 1000]$, and $[1, 10000]$. Ten instances are generated for each of the 39 classes defined by each combination of m, n and processing time interval.

The two families differ by the distribution of the processing times. The instances in the first family were generated by França et al. [10] with processing times uniformly distributed in each interval. The generator developed by Frangioni et al. [11] for the second family was obtained from [23]. For a given interval $[a, b]$ of processing times, with $a = 1$ and $b \in \{100, 1000, 10000\}$, their generator selects 98% of the processing times from a uniform distribution in the interval $[0.9(b - a), b]$ and the remaining 2% in the interval $[a, 0.2(b - a)]$.

4.2 Search strategy

In the first computational experiment, we compare three different search methods that can be used with HI_PCmax.

In all three methods the core procedure `C+R+I` is used to check whether there exists a solution with a certain target makespan in the interval $[LB, UB]$. In the lower bound method (LBM), we start the search using LB as the target makespan, which is progressively increased by one. In the binary search method (BSM), the interval $[LB, UB]$ is progressively bisected by setting $\lfloor (LB+UB)/2 \rfloor$ as the target makespan. The binary search with prespecified entry point method (EBS) is that used in the pseudo-code in Figure 2. It follows the same general strategy of BSM, but starts using LB as the first target makespan.

Table 1. Search strategies: LBM, BSM, and EBS.

		Search method								
		LBM			BSM			EBS		
Instances	t_i	time (s)	max	avg	time (s)	max	avg	time (s)	max	avg
uniform	$[1, 100]$	0.17	1	1.00	0.17	1	1.00	0.14	1	1.00
	$[1, 1000]$	3.02	14	2.88	2.16	6	3.38	1.98	6	2.38
	$[1, 10000]$	33.42	190	1.71	11.77	10	6.50	11.67	10	6.23
non-uniform	$[1, 100]$	16.07	8	1.73	21.56	5	3.56	14.02	4	1.47
	$[1, 1000]$	47.08	26	1.77	54.14	9	6.06	21.94	6	1.15
	$[1, 10000]$	499.77	254	9.18	491.21	12	8.98	83.11	10	2.02

Table 1 presents the results observed with each search method. For each family of test problems, we report the following statistics for the 130 instances with the same interval for the processing times: the total computation time in seconds, the maximum and the average number of executions of the core procedure `C+R+I`. These results show that EBS is consistently better than the other methods: the same solutions are always found by the three methods, with significantly smaller total times and fewer executions of `C+R+I` in the case of EBS.

4.3 Phases

In this experiment, we investigate the effect of the preprocessing, construction, redistribution, and improvement phases. Four variants of the hybrid improvement procedure HI_PCmax are created:

- Variant `P`: only lines 1–4 corresponding to the preprocessing phase of the pseudo-code in Figure 2 are executed.
- Variant `P+C`: the core procedure `C+R+I` is implemented without the redistribution and improvement phases.
- Variant `P+C+R`: the core procedure `C+R+I` is implemented without the improvement phase.
- Variant `P+C+R+I`: the core procedure `C+R+I` is fully implemented with all phases, corresponding to the complete HI_PCmax procedure itself.

Table 2 shows the results obtained with each variant. The differences between corresponding columns associated with consecutive variants give a picture of the effect of each additional phase. For each family of test poblems and for each interval of processing times, we report the number of optimal solutions found and the total computation time in seconds over all 130 instances.

These results show that the uniform instances are relatively easy and the construction, redistribution, and improvement phases do not bring significative benefits in terms of solution quality or computation times. We notice that 90.5% of the 390 uniform instances are already solved to optimality after the preprocessing phase. The three additional phases allow solving only 13 other instances

Table 2. Phases: preprocessing, construction, redistribution, and improvement.

		Variants							
		P		C		C+R		C+R+I	
Instances	p_i	opt.	time(s)	opt.	time (s)	opt.	time (s)	opt.	time (s)
uniform	$[1, 100]$	130	0.06	130	0.06	130	0.07	130	0.14
	$[1, 1000]$	122	1.37	123	1.38	125	1.59	126	1.98
	$[1, 10000]$	101	2.50	103	2.77	110	4.70	110	11.67
		353		356		365		366	
non-uniform	$[1, 100]$	71	0.48	71	1.17	85	20.51	120	14.02
	$[1, 1000]$	65	0.62	65	1.90	70	40.76	128	21.94
	$[1, 10000]$	68	1.08	68	4.44	70	87.10	121	83.11
		204		204		225		369	

to optimality, at the cost of multiplying the total computation time by a factor of almost five. This picture is quite different for the non-uniform instances. In this case, only 204 out of the 390 test problems (52.3%) are solved to optimality after the preprocessing phase. The complete procedure with all phases made it possible to solve 165 additional instances to optimality.

In consequence of the order in which the three heuristics are applied in the preprocessing phase (lines 1, 8, and 12 of the pseudo-code in Figure 2), of the 557 optimal solutions found after this phase, 171 were obtained with the LPT heuristic H3, one with the $1/5 + 2^{-k}$-approximation heuristic H2, and 385 with the construction heuristic H1 proposed in Section 2. However, we note that if the lower bound $\max(L_2, L_3, L_\vartheta, L_{HS})$ is used, then heuristic H1 alone is capable of finding 556 out of the 557 optimal solutions obtained during the preprocessing phase. This shows that H1 is indeed a very useful fast heuristic to $P||C_{\max}$.

4.4 Comparison with other approaches

In this final set of experiments, we compare the hybrid improvement heuristic HI_PCmax with the list scheduling algorithm LPT [15], the 3-PHASE heuristic of França et al. [10], and the branch-and-bound algorithm B&B of Dell'Amico and Martello [7] with the number of backtracks set at 4000 as suggested by the authors, as well as with the best solution found by the multi-exchange (ME) algorithms.

Tables 3 to 5 report the results obtained by heuristics LPT, B&B, HI_PCmax, and 3-PHASE for the uniform instances. Tables 6 to 8 give the results obtained by heuristics LPT, B&B, HI_PCmax, and ME for the non-uniform instances. The following statistics over all ten test instances with the same combination of m and n are reported: (a) average relative errors with respect to the best lower bound for algorithms LPT, B&B, and HI_PCmax; (b) average relative errors reported in [10] for algorithm 3-PHASE; (c) average relative errors reported in [12] for the best among the solutions obtained by ME algorithms 1-SPT, 1-BPT, and

K-SPT; (d) number of optimal solutions found by LPT, B&B, and HI_PCmax; (e) average computation times observed for LPT, B&B, and HI_PCmax on a 2.4 GHz AMD XP machine; and (f) average computation times reported in [12] for the best ME algorithm on a 400 MHz Pentium II with 256 Mbytes of RAM memory.

Most uniform instances are easy and can be solved in negligible computation times. Tables 3 to 5 show that HI_PCmax found better solutions than LPT, B&B, and 3-PHASE for all classes of test problems. Only four instances in Table 4 and 20 instances in Table 5 were not solved to optimality by HI_PCmax. B&B outperformed LPT and 3-PHASE, but found slightly fewer optimal solutions and consequently slightly larger relative errors than HI_PCmax. We also notice that the uniform test instances get harder when the range of the processing times increase.

The non-uniform test instances are clearly more difficult that the uniform. Once again, HI_PCmax outperformed the other algorithms considered in Tables 6 to 8 in terms of solution quality and computation times. This conclusion is particularly true if one compares the results observed for the largest test instances with $m = 25$ and $n \geq 250$.

Table 9 summarizes the main results obtained by algorithms HI_PCmax and B&B on the same computational environment. For each group of test problems and for each algorithm, it indicates the number of optimal solutions found over the 130 instances, the average and maximum absolute errors, the average and maximum relative errors, and he average and maximum computation times. The superiority of HI_PCmax is clear for the non-uniform instances. It not only found better solutions, but also in smaller computation times.

Table 3. Comparative results, uniform instances, $t_i \in [1, 100]$.

		LPT		B&B			HI_PCmax			3-PHASE
m	n	error	opt	error	opt	time (s)	error	opt	time (s)	error
5	10	3.54e-03	9	0	10	-	0	10	-	0.018
5	50	4.58e-03	1	0	10	-	0	10	-	0.000
5	100	8.81e-04	4	0	10	-	0	10	-	0.000
5	500	0	10	0	10	-	0	10	-	0.000
5	1000	0	10	0	10	-	0	10	-	0.000
10	50	1.56e-02	0	0	10	-	0	10	-	0.002
10	100	3.64e-03	1	0	10	-	0	10	-	0.002
10	500	1.20e-04	7	0	10	-	0	10	-	0.000
10	1000	0	10	0	10	-	0	10	-	0.000
25	50	8.61e-03	6	0	10	-	0	10	0.01	0.011
25	100	2.37e-02	0	0	10	-	0	10	-	0.003
25	500	9.04e-04	4	0	10	-	0	10	-	0.000
25	1000	0	10	0	10	-	0	10	-	0.000

'-' is used to indicate negligible computation times.

Table 4. Comparative results, uniform instances, $t_i \in [1, 1000]$.

		LPT		B&B			HI_PCmax			3-PHASE
m	n	error	opt	error	opt	time (s)	error	opt	time (s)	error
5	10	0	10	0	10	-	0	10	-	0.010
5	50	3.33e-03	0	0	10	-	0	10	-	0.001
5	100	1.02e-03	0	0	10	-	0	10	-	0.000
5	500	4.63e-05	1	0	10	-	0	10	0.02	0.000
5	1000	7.97e-06	5	0	10	-	0	10	0.05	0.000
10	50	1.61e-02	0	8.17e-05	8	0.01	7.74e-05	8	0.02	0.002
10	100	4.04e-03	0	0	10	-	0	10	-	0.000
10	500	2.21e-04	0	0	10	-	0	10	0.01	0.000
10	1000	4.42e-05	3	0	10	-	0	10	0.04	0.000
25	50	1.06e-02	4	0	10	-	0	10	0.02	0.011
25	100	3.15e-02	0	2.07e-04	8	0.11	9.93e-05	8	0.04	0.003
25	500	1.41e-03	0	0	10	-	0	10	-	0.000
25	1000	2.75e-04	0	0	10	-	0	10	0.01	0.000

'-' is used to indicate negligible computation times.

Table 5. Comparative results, uniform instances, $t_i \in [1, 10000]$.

		LPT		B&B			HI_PCmax			3-PHASE
m	n	error	opt	error	opt	time (s)	error	opt	time (s)	error
5	10	6.48e-03	9	0	10	-	0	10	0.03	0.013
5	50	5.92e-03	0	9.26e-06	7	0.01	0	10	0.03	0.000
5	100	1.41e-03	0	0	10	-	0	10	-	0.000
5	500	4.87e-05	0	0	10	-	0	10	0.01	0.000
5	1000	1.02e-05	0	0	10	-	0	10	0.14	0.000
10	50	2.45e-02	0	1.14e-03	0	0.18	1.03e-04	0	0.25	0.004
10	100	4.82e-03	0	0	10	-	0	10	-	0.000
10	500	2.34e-04	0	0	10	-	0	10	0.01	0.000
10	1000	6.31e-05	0	0	10	-	0	10	0.04	0.000
25	50	7.25e-03	6	0	10	-	0	10	0.03	0.004
25	100	2.76e-02	0	4.49e-03	0	0.74	3.47e-04	0	0.59	0.001
25	500	1.00e-03	0	0	10	-	0	10	0.01	[a]
25	1000	3.12e-04	0	0	10	-	0	10	0.02	0.000

'-' is used to indicate negligible computation times.

[a] not reported in [10].

Table 6. Comparative results, non-uniform instances, $t_i \in [1, 100]$.

		LPT		B&B			HI_PCmax			ME	
m	n	error	opt	error	opt	time (s)	error	opt	time (s)	error	time (s)
5	10	0	10	0	10	-	0	10	-	0	-
5	50	9.37e-03	0	7.24e-03	0	0.11	0	10	0.03	8.58e-03	0.01
5	100	1.67e-02	0	0	10	-	0	10	0.01	5.31e-05	0.02
5	500	5.86e-04	0	0	10	0.01	0	10	0.01	0	1.01
5	1000	1.44e-04	0	0	10	-	0	10	0.02	0	9.71
10	50	1.13e-02	0	8.34e-03	2	0.23	7.28e-03	4	0.30	1.57e-02	-
10	100	9.02e-03	0	7.96e-03	0	0.21	2.13e-04	8	0.22	5.09e-03	0.04
10	500	1.00e-02	0	0	10	0.02	0	10	-	2.13e-05	3.26
10	1000	3.83e-04	1	0	10	0.01	0	10	-	0	17.14
25	50	0	10	0	10	-	0	10	-	0	0.01
25	100	4.77e-03	0	2.65e-03	3	1.12	1.34e-03	8	0.55	9.85e-03	0.04
25	500	9.79e-03	0	1.60e-04	8	1.63	0	10	0.08	2.12e-04	3.29
25	1000	9.30e-03	0	1.17e-03	4	9.56	0	10	0.18	7.97e-05	36.59

'-' is used to indicate negligible computation times.

Table 7. Comparative results, non-uniform instances, $t_i \in [1, 1000]$.

		LPT		B&B			HI_PCmax			ME	
m	n	error	opt	error	opt	time (s)	error	opt	time (s)	error	time (s)
5	10	0	10	0	10	-	0	10	-	0	-
5	50	8.82e-03	0	7.94e-03	0	0.20	0	10	0.03	8.92e-03	0.01
5	100	1.70e-02	0	1.55e-04	9	0.02	0	10	0.02	7.43e-05	0.06
5	500	6.35e-04	0	0	10	-	0	10	-	0	1.39
5	1000	1.78e-04	0	0	10	-	0	10	0.02	0	14.80
10	50	4.08e-03	0	1.57e-03	0	0.42	0	10	-	1.46e-02	0.01
10	100	8.66e-03	0	8.18e-03	0	0.39	0	10	0.35	4.64e-03	0.07
10	500	1.01e-02	0	0	10	0.01	0	10	0.07	0	5.61
10	1000	4.12e-04	0	0	10	0.11	0	10	0.06	0	14.61
25	50	0	10	0	10	-	0	10	-	0	0.01
25	100	4.67e-03	0	3.80e-03	0	1.06	1.34e-03	8	1.08	7.93e-03	0.08
25	500	1.00e-02	0	1.39e-03	1	5.53	0	10	0.13	4.25e-05	15.39
25	1000	9.38e-03	0	7.97e-06	9	0.73	0	10	0.43	7.97e-06	138.21

'-' is used to indicate negligible computation times.

Table 8. Comparative results, non-uniform instances, $t_i \in [1, 10000]$.

		LPT		B&B			HI_PCmax			ME	
m	n	error	opt	error	opt	time (s)	error	opt	time (s)	error	time (s)
5	10	0	10	0	10	-	0	10	-	0	-
5	50	8.79e-03	0	8.11e-03	0	0.28	0	10	0.02	8.95e-03	0.01
5	100	1.70e-02	0	1.00e-04	8	0.05	0	10	0.01	5.78e-05	0.09
5	500	6.42e-04	0	0	10	-	0	10	-	0	1.97
5	1000	1.78e-04	0	0	10	-	0	10	0.03	0	13.88
10	50	4.12e-03	0	2.02e-03	0	0.54	4.22e-06	8	0.35	1.46e-02	0.01
10	100	8.61e-03	0	8.28e-03	0	0.52	0	10	0.24	4.63e-03	0.15
10	500	1.02e-02	0	0	10	0.02	0	10	0.01	1.06e+00	7.99
10	1000	4.10e-04	0	0	10	0.01	0	10	0.02	0	15.57
25	50	0	10	0	10	-	0	10	-	0	0.01
25	100	4.73e-03	0	4.16e-03	0	1.34	1.35e-03	3	4.29	7.76e-03	0.14
25	500	1.01e-02	0	7.23e-04	1	12.70	0	10	3.13	1.91e-05	20.37
25	1000	9.40e-03	0	1.86e-06	8	5.55	0	10	0.22	5.31e-06	195.88

'-' is used to indicate negligible computation times.

Table 9. Comparative results: HI_PCmax vs. B&B.

		HI_PCmax						
		opt	absolute error		relative error		time (s)	
Instances	$t_i \in$		avg	max	avg	max	avg	max
uniform	$[1, 100]$	130	0.00	0	0	0	0	0.09
	$[1, 1000]$	126	0.03	1	1.36e-05	5.15e-04	0.02	0.19
	$[1, 10000]$	110	0.75	12	3.46e-05	5.83e-04	0.09	0.77
non-uniform	$[1, 100]$	120	0.32	7	6.79e-04	1.50e-02	0.11	3.76
	$[1, 1000]$	128	0.38	25	1.03e-04	6.71e-03	0.17	9.49
	$[1, 10000]$	121	3.90	253	1.04e-04	6.75e-03	0.64	19.62
		B&B						
		opt	absolute error		relative error		time (s)	
Instances	$t_i \in$		avg	max	avg	max	avg	max
uniform	$[1, 100]$	130	0.00	0	0	0	0	0.01
	$[1, 1000]$	126	0.05	3	2.22e-05	1.55e-03	0.01	0.75
	$[1, 10000]$	107	9.31	173	4.33e-04	8.30e-03	0.07	1.12
non-uniform	$[1, 100]$	87	1.84	20	2.12e-03	1.50e-02	0.99	35.41
	$[1, 1000]$	79	15.59	152	1.77e-03	9.42e-03	0.65	9.99
	$[1, 10000]$	77	150.01	880	1.80e-03	9.73e-03	1.62	23.60

5 Concluding Remarks

We proposed a new strategy for solving the multiprocessor scheduling problem, based on the application of a hybrid improvement heuristic to the bin packing problem. We also presented a new, quick construction heuristic, combining the LPT rule with the solution of subset sum problems.

The construction heuristic revealed itself as a very effective approximate algorithm and found optimal solutions for a large number of test problems. The improvement heuristic outperformed the other approximate algorithms in the literature, in terms of solution quality and computation times. The computational results are particularly good in the case of non-uniform test instances.

Acknowledgments: The authors are grateful to M. Dell'Amico for having kindly provided the code of B&B algorithm used in the computational experiments. We are also thankful to P. França for making available the instances of the uniform family.

References

1. A.C.F. Alvim, C.C. Ribeiro, F. Glover, and D.J. Aloise, "A hybrid improvement heuristic for the one-dimensional bin packing problem", *Journal of Heuristics*, 2004, to appear.
2. A.C.F. Alvim, *Uma heurística híbrida de melhoria para o problema de bin packing e sua aplicação ao problema de escalonamento de tarefas*, Doctorate thesis, Catholic University of Rio de Janeiro, Department of Computer Science, Rio de Janeiro, 2003.
3. J. Błażewicz, "Selected topics in scheduling theory", in *Surveys in Combinatorial Optimization* (G. Laporte, S. Martello, M. Minoux, and C.C. Ribeiro, eds.), pages 1–60, North-Holland, 1987.
4. J.L. Bruno, E.G. Coffman Jr., and R. Sethi, "Scheduling independent tasks to reduce mean finishing time", *Communications of the ACM* 17 (1974), 382–387.
5. T. Cheng and C. Sin, "A state-of-the-art review of parallel-machine scheduling research", *European Journal of Operational Research* 47 (1990), 271–292.
6. E.G. Coffman Jr., M.R. Garey, and D.S. Johnson, "An application of bin-packing to multiprocessor scheduling", *SIAM Journal on Computing* 7 (1978), 1–17.
7. M. Dell'Amico and S. Martello, "Optimal scheduling of tasks on identical parallel processors", *ORSA Journal on Computing* 7 (1995), 191–200.
8. S.M. Fatemi-Ghomi and F. Jolai-Ghazvini, "A pairwise interchange algorithm for parallel machine scheduling", *Production Planning and Control* 9 (1998), 685–689.
9. G. Finn and E. Horowitz, "A linear time approximation algorithm for multiprocessor scheduling", *BIT* 19 (1979), 312–320.
10. P.M. França, M. Gendreau, G. Laporte, and F.M. Müller, "A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective", *Computers Ops. Research* 21 (1994), 205–210.
11. A. Frangioni, M. G. Scutellà, and E. Necciari, "Multi-exchange algorithms for the minimum makespan machine scheduling problem", Report TR-99-22, Dipartimento di Informatica, Università di Pisa, Pisa, 1999.

12. A. Frangioni, E. Necciari, and M. G. Scutellà, "A multi-exchange neighborhood for minimum makespan machine scheduling problems", *Journal of Combinatorial Optimization*, to appear.
13. D.K. Friesen, "Tighter bounds for the MULTIFIT processor scheduling algorithm", *SIAM Journal on Computing* 13 (1984), 170–181.
14. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.
15. R.L. Graham, "Bounds on multiprocessing timing anomalies", *SIAM Journal of Applied Mathematics* 17 (1969),416–429.
16. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey", *Annals of Discrete Mathematics* 5 (1979), 287–326.
17. D.S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems: Theoretical and practical results", *Journal of the ACM* 34 (1987), 144–162.
18. R. Hübscher and F. Glover, "Applying tabu search with influential diversification to multiprocessor scheduling", *Computers and Operations Research* 21 (1994), 877–884.
19. D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham, "Worst case performance bounds for simple one-dimensional packing algorithms", *SIAM Journal on Computing* 3 (1974), 299–325.
20. E.L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, "Sequencing and scheduling: Algorithms and complexity", in *Logistics of Production and Inventory: Handbooks in Operations Research and Management Science* (S.C. Graves, P.H. Zipkin, and A.H.G. Rinnooy Kan, eds.), 445–522, North-Holland, 1993.
21. S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, 1990.
22. McNaughton, "Scheduling with deadlines and loss functions", *Management Science* 6 (1959), 1–12.
23. E. Necciari,"Istances of machine scheduling problems". Online document available at `http://www.di.unipi.it/di/groups/optimize/Data/MS.html`, last visited on November 21, 2001.
24. A. Scholl and S. Voss, "Simple assembly line balancing - Heuristic approaches",*Journal of Heuristics* 2 (1996), 217–244.
25. M. Yue, "On the exact upper bound for the MULTIFIT processor scheduling algorithm", *Annals of Operations Research* 24 (1990), 233–259.