A Hybrid Heuristic for Bin-Packing and Multiprocessor Scheduling

Adriana C.F. Alvim*

August 27, 2010

1 Introduction

The bin packing (BP) problem consists of finding the minimum number m of bins of capacity C necessary to pack a set of weighted items so that the sum of the weights of the items in each bin does not exceed the bin's capacity. BP is known to be closely related to the multiprocessor scheduling problem ($P\|C_{\max}$), which is the problem of scheduling n independent tasks with associated processing times w_i on m parallel identical processors with the objective of minimizing the maximum completion time of a task (makespan). Both problems are NP-hard [6]. They share the same decision problem, which is to determine whether all items/tasks can be assigned to m bins/processors with bin capacity/makespan equal to C. This kind of dual relationship is explored in the works of Alvim et al. [3, 2].

The authors present a hybrid improvement method incorporating tabu search for the bin packing problem (HI_BP) and its associated multiprocessor scheduling problem (HI_PCmax). The overall method consists of a suite of component procedures, each having a specific preprocessing step and sharing the same core procedure formed by the construction, redistribution and improvement phases. Tabu search is at the heart of the improvement phase, fulfilling the function of restoring solutions to feasibility each time a construction phase produces a solution that violates the problem constraints.

A solution is feasible to BP if the makespan is no greater than the bin capacity. A solution is feasible to $P\|C_{\max}$ if it uses no more than m bins. The construction phase builds a feasible solution to $P\|C_{\max}$. Whenever a

*Department of Applied Informatics, Federal University of the State of Rio de Janeiro, Avenida Pasteur 458, Rio de Janeiro, RJ, 22.290-240, Brazil. E-mail: adriana@unriotec.br

feasible solution to $P||C_{\max}$ is not feasible to BP, redistribution phase apply load balancing/unbalancing strategies to improve bin usability. Finally, if the current solution is not feasible to BP, the improvement phase uses tabu search as a means to eliminate capacity violations.

2 The Tabu Search Strategy

The tabu search strategy for reducing capacity violations in the current infeasible solution operates as follows. The search neighborhood consists of swap moves which exchange pairs of items, one of them always from a *violated* bin. Each move $i \leftrightarrow k$ is defined by an ordered pair (i, k) of items from different bins. The first element in the pair is always an item in the target *violated* bin, whose excess deviation is to be reduced. Only moves that decrease the excess deviation of the target violated bin are considered, i.e., swap moves for which $w_i > w_k$. Let $\Delta(i, k)$ be the value of the excess violation associated exclusively with the bins where these items are placed after their exchange. Six types of moves are considered, one for each possible combination of the bins situation after a move $i \leftrightarrow k$: (1) *complete* and *complete*, (2) *complete* and *incomplete*, (3) *incomplete* and *incomplete*, (4) *complete* and *violated*, (5) *incomplete* and *violated* and (6) *violated* and *violated*. The rules allow different choices at different iterations, and are particularly useful in the context of moves leading to infeasible solutions that may eventually be made feasible at a later step.

The proposed strategy may be seen as a variant of the "persistent vote" approach [7, page 134] which also embodies an oscillation strategy. The integration of multiple decision rules criterion, suggested by the vote, gives a useful basis for better choices.

One of the important aspects of the oscillation strategy is that it provides a dynamic variation among the elements subjected to the oscillation, and gives a basis for learning strategies to enhance the form of the oscillation.

The tabu search strategy known as logical restructuring, which is based on anticipatory analysis [7], is used to enhance the efficacy of these steps. In this context, the logical restructuring tries to answer the following questions: "Which conditions assure the existence of a path that leads to a better solution?" and "Which intermediate moves can be created to permit these conditions?". Intermediate moves are then generated by modifying the evaluation used to select transition between solutions or modifying neighborhood structure which determine these transitions [7].

3 Experiments and remarks

We report computational experiments for BP and for $P\|C_{\max}$. Experiments were performed on a 1.7 GHz Pentium IV, 256 Mbytes of RAM memory. Detailed computational results are available at [1]. We consider four groups of test problems for BP. Group-I, distributed by the OR Library (http://mscmga.ms.ic.ac.uk/jeb/orlib/binpackinfo.html), has 160 instances. Group-II is composed by 1210 instances available from Scholl and Klein [10]. Group-III is composed by 217 instances distributed by SICUP [13] and Group-IV is composed by 100 instances of each of 145 ffd-hard and extremely-ffd-hard classes proposed by Schwerin and Wäscher [11]. HI_BP found optimal solutions for all instances of Group-I and Group-II. HI_BP also solved four open instances from Group-II, as reported by Scholl and Klein [10]. HI_BP improved the best results for 11 instances of Group-III and found the same results for the others. We also compared our results with those obtained by MTPCS [12] for Group-IV. HI_BP performed consistently better than MTPCS, solving to optimality 97.9% of the 14500 instances against 84.2% by MTPCS.

We considered two groups of test problems for $P\|C_{\max}$. Each group is formed by ten test problems for each of 39 classes. These classes are characterized by different combinations of $m \in \{5, 10, 25\}$ and $n \in \{10, 50, 100, 500, 1000\}$ with processing times randomly generated in the intervals [1,100], [1,1000], and [1,10000]. The two groups differ by the distribution of the processing times: uniform [5] and non-uniform [8]. We compared HI_PCmax [2] with the branch-and-bound code (B&B) by Dell'Amico and Martello [4] with the number of backtracks set at 4000. Table 1 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 maximum absolute errors (w.r.t. the best lower bound), the average relative errors, and the average computation times in seconds. The superiority of HI_PCmax is clear for the non-uniform instances. It not only found better solutions, but also in smaller computation times. Recently work of Paletta and Vocaturo [9] compared their composite algorithm (CA) with our HI_PCmax for the same set of $P\|C_{\max}$ test problems and also for unpublished results for Group-I of bin packing instances. The results obtained by CA and HI_PCmax are comparable, both in quality and running time.

The move selection strategy used by the Tabu Search, which combines a variant of persistent vote and anticipatory analysis, is a major contribution

Table 1: Comparative results: HI_PCmax vs. B&B.

		HI_PCmax				B&B			
		opt	max abs	rel avg	time	opt	max abs	rel avg	time
Group	$t_i \in$		error	error	(s)		error	error	(s)
uniform	$[1, 100]$	130	0	0.0000	0	130	0	0.0000	0
	$[1, 1000]$	126	1	1.36e-05	0.02	126	3	2.22e-05	0.03
	$[1, 10000]$	110	12	3.46e-05	0.15	107	173	4.33e-04	0.17
non-uniform	$[1, 100]$	120	7	6.79e-04	0.14	87	20	2.12e-03	2.12
	$[1, 1000]$	128	25	1.03e-04	0.20	79	152	1.77e-03	1.52
	$[1, 10000]$	121	253	1.04e-04	0.72	77	880	1.80e-03	3.99

of this work and very likely can be applied to other problems in similar situations. HI_BP and HI_PCmax compare favorably with other approaches. In addition to dominating other methods when it was published more than five years ago, recently published work [9] for the $P||C_{\max}$ shows that HI_PCmax is still a competitor in terms of solution quality and/or computation times. Observations from [3] about potential enhancements that may further improve the basic method remain to be investigated.

References

[1] A. C. F. Alvim. Publications. `http://www.uniriotec.br/~adriana/publications.html`.

[2] A. C. F. Alvim and C. C. Ribeiro. A hybrid bin-packing heuristic to multiprocessor scheduling. In C.C. Ribeiro and S.L. Martins, editors, *Lecture Notes in Computer Science*, volume 3059, pages 1–13. Springer-Verlag, 2004.

[3] 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*, 10:205–229, 2004.

[4] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200, 1995.

[5] 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 and Operations Research*, 21:205–210, 1994.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[7] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[8] E. Necciari, 2001. http://www.di.unipi.it/di/groups/optimize/Data/MS.html.

[9] Giuseppe Paletta and Francesca Vocaturo. A composite algorithm for multiprocessor scheduling. *Journal of Heuristics*, pages 1–21, 2010. 10.1007/s10732-010-9135-1.

[10] A. Scholl and R. Klein, 2003. http://www.wiwi.uni-jena.de/Entscheidung/binpp/.

[11] P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4:337–389, 1997.

[12] P. Schwerin and G. Wäscher. A new lower bound for the bin-packing problem and its integration to MTP. *Pesquisa Operacional*, 19:111–129, 1999.

[13] SICUP, 2003. http://www.apdio.pt/sicup/Sicuphomepage/research.htm.