Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling*

Roland Hübscher
Department of Computer Science & Institute of Cognitive Science
University of Colorado at Boulder
Boulder, CO 80309-0430

Fred Glover
Graduate School of Business Administration
University of Colorado at Boulder
Boulder, CO 80309-0419

Abstract

We describe a tabu search approach to the scheduling problem of minimizing the makespan on n tasks on m equivalent processors. This problem is isomorphic to a variant of the multiple bin packing problem. We make use of a candidate list strategy that generates only a small subset of all possible moves, and employ a dynamic tabu list for handling tabu restrictions. We also introduce an influential diversification component to overcome an entrenched regionality phenomenon that represents a "higher order" difficulty encountered by local search methods. Influential diversification notably improves the behavior and quality of the solutions of our tabu search procedure as the search horizon grows. Results are presented for a range of problems of varying dimensions, and our method is also compared to an extended simulated annealing approach that previously has produced the best solutions for the isomorphic bin packing problem.

1 Introduction

Minimizing the makespan on multiprocessors is hard for local optimization techniques, because neighboring solutions differ widely in quality. Pure simulated annealing does not seem to overcome this problem as Kämpke (1988) and Johnson *et al.* (1991) report on bin-packing problems similar to the scheduling problem. We will focus our attention on ways to escape not only deep local minima but also to overcome an *entrenched regionality* phenomenon that combinatorial optimization problems of this type are prone to exhibit. We report computational outcomes for a variety of problems with different numbers of tasks and processors, and also address a classical 10 bin/100 item problem from an isomorphic Minmax bin packing formulation, demonstrating our approach obtains solutions that significantly improve the previous best known outcome for this problem.

The multiprocessor scheduling problem may be defined as follows. Let $L = \{q_1, q_2, \ldots, q_n\}$ denote a collection of n tasks which must be assigned to m sets $P_1, P_2, \ldots, P_m$, thereby creating a

*Published in *Computers & Operations Research*, Vol. 21, No. 8, pp. 877–884.

partition of L. Each task set P_i is assigned to processor i, one of the m equivalent processors. The goal is to find an assignment that minimizes the makespan of the m parallel processors, that is, to find

$$\min \max_{1 \le i \le m} \sum_{q_j \in P_i} t(q_j),$$

where $t(q_j)$ is the time it takes to execute task q_j on any of the m equivalent processors. The value $T_i = \sum_{q_j \in P_i} t(q_j)$ is the latest task finishing time of processor i, and the goal may be expressed as that of distributing the tasks into $P_1, \ldots, P_m$ to minimize $\max_i T_i$.

This scheduling problem can also be viewed as assigning n items $q_1, \ldots, q_n$ with weight $t(q_i)$ to m bins with the goal of minimizing the weight of the heaviest bin. In this case, T_i can be interpreted as the weight of bin i.

2 Tabu Search

The main goal of our approach is to find how local search methods can be extended to do well in search spaces where certain types of solution structures tend to become locked into place, creating a form of regional entrenchment that prevents access to solutions of superior quality. Our solution to this problem is twofold and consists of a dynamic tabu list with a moving gap coupled with influential (strategically composed) diversification moves that are executed rarely. The fundamental ideas and procedural considerations of tabu search have been described in a number of references (see, e.g., Glover and Laguna (1992), for a review of recent developments and applications). The following description will therefore concentrate on the features that distinguish the present approach from others, in particular by characterizing the dynamics of the tabu list and the influential diversification component.

We employ a simple neighborhood for defining moves (transitions between solutions) that consists of exchanging two tasks between two processors. Similar types of exchange moves have been applied to combinatorial problems in a variety of other settings. (See, for example, de Werra and Hertz (1989), Skorin-Kapov (1990), and Weber and Liebling (1986).) An initial solution is constructed using the best-fit random-order heuristic. That is, initially the tasks are shuffled, and are then successively assigned by selecting at each step a machine whose currently assigned tasks consume the least total processing time.

In overview, following standard tabu search methodology, each iteration generates a set of candidate moves from the total collection of moves in the neighborhood by a filtering process. The goal of the filtering is to assure, subject to tradeoffs in the effort of generating and examining these moves, that the candidate set contains moves with the *highest* evaluations. The candidate set is further screened to retain only the candidates that qualify as *admissible*, that is, those candidates that are not tabu or that fulfill the aspiration criteria. A candidate is tabu if it is characterized by a predicate on the tabu list. The tabu list reflects the recent move history of the search and

implements intensification and diversification strategies (which respectively focus on reinforcing attributes of attractive solutions and on driving the search into new regions). The admissible candidate with the highest evaluation is then selected, generating the associated new solution, and the tabu restrictions and aspiration criteria are updated.

2.1 Move Generation

A simple average case analysis shows there are about n/m tasks assigned to each processor, which allows $(n/m)^2 \times m^2/2 = n^2/2$ possible exchange moves. The goal is to generate a small candidate list of alternatives that are likely to be superior. To do this, it is necessary to define what qualifies a candidate as being good.

We represent a move that exchanges a task $q_i \in P_i$ with a task $q_j \in P_j$ by the notation $(P_i, q_i) \leftrightarrow (P_j, q_j)$.[1] By convention, the processor that currently has the latest task finishing time is on the left side of the $\leftrightarrow$ sign, as identified by processor i in our case, and the other processor, j, satisfies $T_j < T^*$, where $T^* = \sum_{q_i \in L} t(q_i)/m$ is the ideal last task finishing time (the target time for each processor if an ideal solution were possible).

We include the special case where one of the tasks is a dummy task that takes zero time units to execute, thereby incorporating partial exchanges that simply move a task from one processor to another. Since the dummy task is never explicitly moved, there is no loss of efficiency. However, this convention considerably simplifies the generation and evaluation of the candidates and their implementation.

Evaluation

The value of a move is defined to be the change of the mathematical variance of the last task finishing time of the processors, given by

$$\begin{aligned} v((P_i, q_i) \leftrightarrow (P_j, q_j)) \quad = \quad & (T^* - T_i)^2 - (T^* - (T_i - t(p_i) + t(p_j)))^2 + \\ & (T^* - T_j)^2 - (T^* - (T_j - t(p_j) - t(p_i)))^2 \end{aligned}$$

Our adoption of a measure of goodness based on variance is motivated by the intensification theme in tabu search, following the approach used in (Glover and McMillan, 1986). We seek to accentuate the search focus on sets that are farther removed from the ideal state and reduce the discrepancy from this state. A further way to reduce the set of candidates is to restrict moves to those exchanges that occur only between the processor with the latest finishing time and the processors i with $T_i < T^*$. This constraint and the characteristics of the goodness measure allow a fast generation of a small set of superior candidates that contains only $cn/2$ potential moves for a small constant c.

[1] We use the same indices for tasks and task sets as often as possible to simplify the association between these entities.

Execution

To execute this measure efficiently, a special organization scheme is used that identifies an ideal weight for an item to be exchanged with a given item. In particular, given two sets P_i, P_j and a task q_i in P_i, we identify the ideal makespan t_o for a task in P_j if it is to be exchanged with q_i. This is achieved by solving $\partial v((P_i, q_i) \leftrightarrow (P_j, q_o))/\partial t(q_o) = 0$, which gives $t_o = t(q_i) + (T_j - T_i)/2$. Furthermore, $v((P_i, q_i) \leftrightarrow (P_j, q_j)) > v((P_i, q_i) \leftrightarrow (P_j, q_k))$ if $|t_o - t(q_j)| < |t_o - t(q_k)|$, where $q_j, q_k \in P_j$ and t_o is the ideal weight computed as shown above. Thus, the candidate moves $(P_i, q_i) \leftrightarrow (P_j, q_j)$ can be ordered according to increasing values of $|t(p_i) + (T_j - T_i)/2 - t(p_j)|$, and we say that move $(P_i, q_i) \leftrightarrow (P_j, q_j)$ is a level l candidate with respect to P_i, q_i, and P_j if q_j is the l^{th} best choice in set P_j with respect to the indicated values.

Let $P_L = \{i : T_i < T^*\}$ be the set of indices of all task sets with lower than average makespan and let h be the index of a set with greatest makespan, that is, $T_h \geq T_i$, $1 \leq i \leq m$. The set of level l candidates then is defined by $C_l = \{(P_h, q_h) \leftrightarrow (P_j, q_j) : j \in P_L$ and $(P_h, q_h) \leftrightarrow (P_j, q_j)$ is a level l candidate$\}$. By convention, the first set in a move description always has the longest makespan and the second set of the move description has a below average makespan.

To generate a small number of possible candidates, we proceed as follows. First, C_1 is generated and if an admissible candidate is found the generation stops; otherwise, C_2 is generated and so on. Since the maximal level for generation is a fixed number it may happen that no admissible candidate is found; in this case the search for an admissible candidate stops and a "least inadmissible" candidate is selected, which is the move. Using a balanced tree to store the items in a set, the level 1 candidate can be found in log-time per task set. Finding level l candidates, $l > 1$, requires only a constant number of steps if the items are sorted with respect to their makespan. This ordering is advantageous since $v((P_i, q_i) \leftrightarrow (P_j, q_j)) > v((P_i, q_i) \leftrightarrow (P_j, q_k))$ if $|t(q_o) - t(q_j)| < |t(q_o) - t(q_k)|$. A simple average case analysis shows that $n/2$ candidates are generated per level and that generating l levels requires about $(l - 1 + \log n/m)n/2$ steps.

To determine the efficacy of these ideas, we tested the candidate list approach against the alternative of considering all possible moves at each iteration, running each approach for the same number of iterations, on preliminary test problems. The outcome showed that the use of the candidate list did not cause solution quality to deteriorate, disclosing that the indicated process indeed generates an effective subset of candidates.

2.2 Dynamic Tabu List

A dynamic tabu list of changing size and composition is applied in our procedure to integrate intensification and diversification strategies. The dynamic tabu list overcomes the difficulty of getting stuck in repetitions of potentially long sequences of moves. A tabu list can be viewed as a list of predicates. If a predicate in the tabu list applies to a candidate move, the candidate is tabu, that is, the move cannot be chosen unless it fulfills the aspiration criteria. A candidate that may

be chosen is called admissible and the best admissible candidate is finally executed.

Tabu Criteria

Since every candidate must be checked against the tabu criteria, the tests should not only filter out the right candidates but should also be easy and fast to compute. Whether the candidate move $(P_i, q_i) \leftrightarrow (P_j, q_j)$ is tabu depends on the sets of tasks P_i, P_j and length $t(q_i), t(q_j)$ of the two tasks to be exchanged, but not on their identity.

A candidate $(P_i, q_i) \leftrightarrow (P_j, q_j)$ is tabu if it fulfills one of the following three conditions. First, a move is automatically excluded from consideration if $t(q_i) = t(q_j)$. Such a move is called a *null move* in tabu search terminology and is forbidden by default. Second, the candidate is tabu if the previous move involved the same two processors but with reversed roles, that is, if $(P_j, q'_j) \leftrightarrow (P_i, q'_i)$ was the previous move for some q'_i, q'_j. In this case, the task lengths do not matter. This criterion is restricted to apply only if there are at least two task sets P_i with $T_i < T^*$.

The third criterion uses the predicates in the tabu list. For each move $(P_a, q_a) \leftrightarrow (P_b, q_b)$ executed at time t_e, the predicates $late(P_a, t(q_a))$ and $early(P_b, t(q_b))$ are stored in slot t_e of the tabu list, that is, the tabu list remembers what task length has been moved *out* of a set. This information is then used to avoid moving a task with same length back into the same set too soon, which forbids "reversing" the move (and some set of related moves) for a chosen duration. The current candidate move $(P_i, q_i) \leftrightarrow (P_j, q_j)$ fulfills the third tabu criterion[2] if $early(P_i, t(q_i))$ and $late(P_j, t(q_j))$ are on the tabu list. This is only a good idea as long as the moves made are of high quality. If a selected move possibly is bad, then preventing its reversal by use of the tabu restrictions can temporarily compel the search to pursue an undesirable course.

These conditions make many more candidates tabu than just the ones that would reverse an earlier move. We do not just exclude a part of the path the search procedure has taken, but a whole region of the search space. However, since the tabu list has a finite length and a moving gap (as identified below), it does not prevent the search from traversing the previously excluded region again, this time coming from a different part of the search space. This allows the search procedure to view a region from different angles, giving it the chance to discover the more promising paths for uncovering good solutions.

Moving Gap

The tabu list consists of a static part and a dynamic part. The configuration of the dynamic part is changed so that an intensification phase is followed by a diversification phase and vice versa. This is done by moving a gap in the tabu list back and forth depending on the requirements of the current situation. This provides an ability to escape from very long repetitions of move sequences,

[2] We give here a slightly simplified version of this criterion.

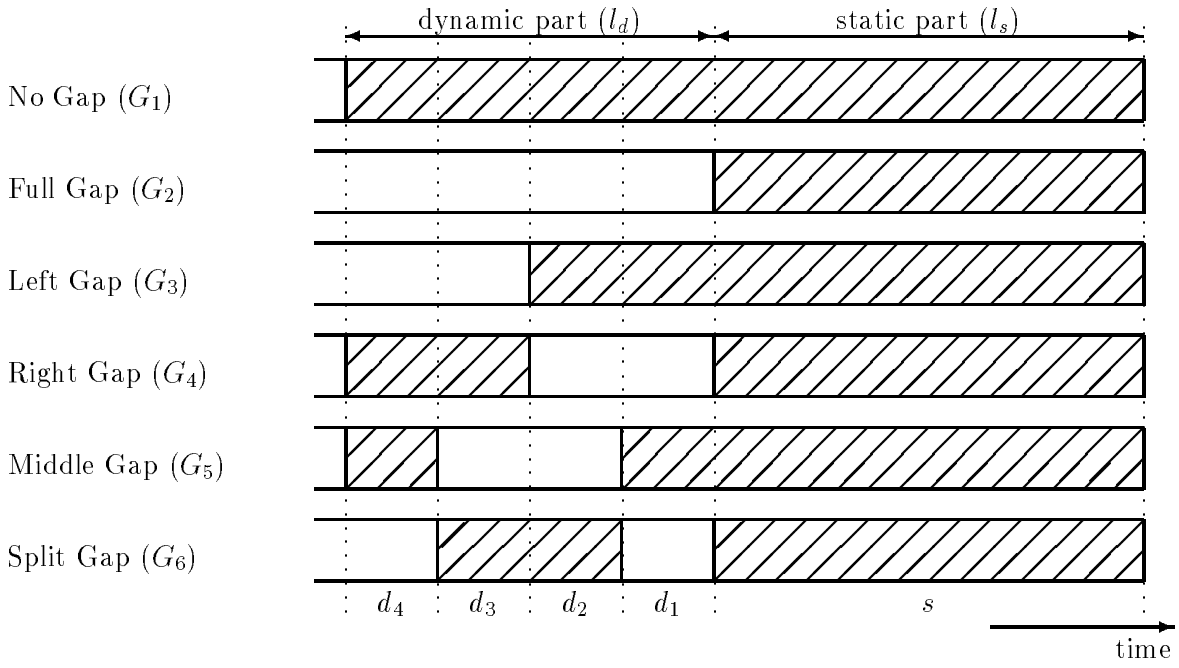

Figure 1: The six configurations of the moving gap tabu list. The hatched parts are tabu.

which often pose a threat to local search methods. However, it does not always get the search to break free of entrenched regionality phenomena.

When a move is executed at time t_e, slot t_e is updated with instantiations of some of the *early* and *late* predicates. The tabu list in our approach consists of a static part s and a dynamic part d as shown in Figure 1. Let t_c be the current time. The static part of the list has l_s slots and stores the predicates P_{t_s}, $t_c - l_s \leq t_s < t_c$. The dynamic part of the list is of length l_d and is partitioned into four equal sublists d_i, $1 \leq i \leq 4$, containing the predicates $P_{t_{d_i}}$, $t_c - l_s - \lfloor id_i/4 \rfloor \leq t_{d_i} < t_c - l_s - \lfloor (i-1)d_i/4 \rfloor$. A configuration G_i of the tabu list is a combination of the static part and some sublists of the dynamic part.

The configuration is changed after a certain number of moves in a manner that alternates even numbered configurations with odd numbered configurations, because the odd numbered configurations G_1, G_3, G_5 tend to diversify the search whereas G_2, G_4, G_6 tend to intensify the search. Generally, the longer the initial part that is tabu (the right most hatched part in Figure 1) the more the search diversifies. Let l be the total number of tabu slots of the current configuration, $l_s \leq l \leq l_s + l_d$. A configuration is changed to its successor with respect to the given sequence when $2l$ moves have elapsed since the last improvement of the best solution found so far. If the search discovers an improved solution it sticks to the same configuration for at least $2l$ moves after the improvement; this is especially useful if the search is intensifying. (Otherwise, the configuration is maintained for $2l$ moves after its initiation.)

Aspiration Criteria

The tabu criteria, if applied unconditionally, sometimes reject worthwhile candidates. To avoid this situation, the aspiration criteria are designed to overrule tabu status and make a tabu candidate $(P_i, q_i) \leftrightarrow (P_j, q_j)$ admissible. We use both a global and a local aspiration criterion. The global condition is fulfilled if the execution of the candidate would result in a new best solution. The local criterion is fulfilled if one of the two involved processors, say i, would get closer to the ideal latest task finishing time than it has at had at any earlier point during the search, without leading to a bad task set for the other processor j.

2.3 Influential Diversification

The tabu list gives our search procedure a big advantage over memoryfree approaches, because it profits from what it has learned (and stored in the tabu list) about neighborhoods encountered during the memory span of the tabu list—which in this case constitutes a form of short term memory. Although the dynamic tabu list and the associated tabu criteria proved to be successful in dealing with repetitions of long move sequences, we found they were sometimes insufficient to guide the search to improve the best solution found. (The approach succeeded in discovering new local optima, but only those from a set dominated by the previous best.) This occurred upon encountering a form of regional "entrenchment" (or imprisonment) which often manifested itself in schedules that had extremely uneven distributions of short and long tasks. Certain processors had many small tasks assigned, and other processors were assigned many big tasks. The simple exchange move used as an underpinning for our approach is not powerful enough to modify the general form of such a structure (although it can produce many different solutions exhibiting this form) because typically all moves that relocate long tasks produce extremely unfavorable changes in the evaluation criteria.

Therefore, when the search gives evidence that regional entrenchment may be in operation, by failing to improve the current best solution for a long period, task sets that contain a relatively large or relatively small number of long tasks are redistributed by an *influential diversification move*. We call the diversification "influential" because it seeks to modify the solution structure in a specific influential (nonrandom) way. (The concept of *move influence* is discussed more fully in (Glover and Laguna, 1992).) The seemingly obvious alternative, which selects two subsets of two tasks each and exchanges them, is much too costly, because there are about $2^{2n/m}m^2/2$ possible moves.[3] Furthermore, our experience suggests that, in appropriate circumstances, a selective approach that focuses on fewer alternatives can result in a behavior that is at least as good as one that exhausts a combinatorially more extensive set of possibilities.[4]

[3]There are about $m^2/2$ pairs of processors and each P_i contains about n/m tasks, that is, $2^{n/m}$ subsets.

[4]The candidate move generation provides an interesting example for this seemingly contradictory proposition. Using the best exchange move from all available alternatives resulted in inferior solutions compared to choosing the best move from a small but strategically screened candidate set as described earlier.

To implement our approach, we first identify the two task sets that have the shortest and the longest tasks. Let $f(i)$ denote the task distribution factor for processor i, which constitutes a normalized sum of the squared task lengths, that is,

$$f(i) = \frac{\sum_{q_j \in P_i} t^2(q_j)}{\left(\sum_{q_j \in P_i} t(q_j)\right)^2} = \sum_{q_j \in P_i} t^2(q_j)/T_i^2.$$

The greater the value of $f(i)$, the greater the number of long tasks P_i contains (approximately speaking). Let i be such that $f(i) \leq f(k)$, $1 \leq k \leq m$, and let j be such that $f(j) \geq f(k)$, $1 \leq k \leq m$, that is, processor i has a deficiency and processor j has an excess of long tasks assigned. We only have to take P_i and P_j into consideration to find a redistribution that has a significant influence on the current solution, creating an altered solution by strategically reallocating their elements.

We now describe a simple but successful strategy for accomplishing this. Let P_i', P_j' be two empty task sets that will be filled as follows. First, take P_j (with an excess of long tasks) and successively assign its tasks, in order of decreasing task length on a best fit basis, to P_i' and P_j'. Then assign elements of P_i in the same way to P_i' and P_j'. Finally replace P_i by P_i' and P_j by P_j', respectively. This procedure forces the longer tasks of P_j into different task sets. Elements of P_i and P_j are distributed between P_i' and P_j' one after the other, because this results in assigning P_i' and P_j' about half of the tasks from P_i and the other half from P_j. This is especially useful if the number of processors m is small.

When is such an influential diversification step executed? Experiments have shown that executing such a step after $3000, 6000, \ldots$ non-improving moves is all that is needed. Shorter intervals sometimes do not allow the search procedure adequate time to home in on a better (local) minimum, especially considering that an influential diversification step results in a significantly altered state for continuing the search.

Of course, more elaborate types of reallocations can be designed to achieve the form of diversification sought. However, the straightforward approach we have indicated, when compared with other strategies, has demonstrated itself to be highly useful. We complete the analysis of this section by giving some reasons why a more advanced method for reallocating tasks on the two selected machines was not able to improve upon our simple approach. (We did not examine reallocations involving larger numbers of machines, given the effectiveness of the approached applied to the two machines singled out.)

One of the more advanced reallocations we tested sought to minimize

$$\left| \left(T_j - \sum_{q_k \in s(T_i)} t\left(q_k\right) \right) - \left(T_i - \sum_{q_k \in s(T_j)} t\left(q_k\right) \right) \right|$$

where $s(T_k)$ is a subsequence of the tasks $q_k \in P_k$ sorted with respect to $t(q_k)$. Additionally, we require $s(T_i)$ and $s(T_j)$ to be greater than 2, because otherwise the move would induce little change.

number of		*average number*	ΔT		
processors	*tasks*	*of iterations*	*average*	*minimum*	*maximum*
2	50	17,607	$1.08\ 10^{-8}$	$9.48\ 10^{-10}$	$1.71\ 10^{-8}$
2	100	19,157	$7.08\ 10^{-10}$	$2.06\ 10^{-11}$	$2.13\ 10^{-9}$
3	100	16,384	$7.52\ 10^{-9}$	$1.17\ 10^{-9}$	$3.34\ 10^{-8}$
3	200	39,293	$6.77\ 10^{-10}$	$4.98\ 10^{-11}$	$1.69\ 10^{-9}$
5	100	36,392	$4.38\ 10^{-8}$	$9.72\ 10^{-9}$	$1.57\ 10^{-7}$
5	200	51,982	$1.38\ 10^{-8}$	$3.97\ 10^{-9}$	$2.90\ 10^{-8}$
10	200	101,801	$4.98\ 10^{-8}$	$2.68\ 10^{-8}$	$8.99\ 10^{-8}$
10	500	84,816	$1.11\ 10^{-8}$	$4.19\ 10^{-9}$	$1.78\ 10^{-8}$
20	500	153,526	$3.19\ 10^{-8}$	$1.59\ 10^{-8}$	$5.89\ 10^{-8}$
20	1000	124,540	$1.76\ 10^{-8}$	$8.59\ 10^{-9}$	$3.46\ 10^{-8}$
50	2000	224,408	$4.53\ 10^{-8}$	$3.24\ 10^{-8}$	$1.00\ 10^{-7}$

Table 1: The results of the first series of test runs. The same tabu list length was used for all problems ($l_s = 12$, $l_d = 9$). The search stopped after 20,000 non-improving moves.

(The definition of $s(T_k)$ allows a relatively fast execution of the move, because it requires only one traversal of each of the two sets P_i and P_j.) However, this approach did not do as well as the one described first. Although seemingly more refined, the strategy does not force long tasks to be sufficiently redistributed, which is an essential aspect of making the outcome qualify as influential. Increasing the frequency of executing these refined moves also did not prove advantageous, perhaps because they induce inadequate perturbations, whereas a major earthquake (of the right character) is needed to free the search procedure from a regional entrenchment. Similar limitations were observed with other "more advanced" strategies.

Our findings suggest that starting with a finely tuned short term memory component of tabu search, and coupling it with a properly designed influential diversification component (that may be quite simple) can produce very effective results—as we now show.

3 Computational Tests and Discussion

To test the efficacy of our approach, we first generated scheduling problems with uniformly distributed task lengths. The relative difference ΔT between the makespan T of the schedule and the ideal schedule length is defined by $\Delta T = (T - T^*)/T^*$.

First, we tested our search procedure without the influential diversification component on randomly generated multiprocessor scheduling problems from 2 to 50 processors and 50 to 2000 tasks subject to the constraint $n \geq 20m$. The same tabu list size has been used in all cases. The task lengths of the randomly generated problems were uniformly distributed in the range [0,1]. Our method was run once on each of the 110 problems and for 38% of the problems found solutions with $\Delta T \leq 10^{-8}$. All remaining solutions except two gave $\Delta T \leq 10^{-7}$, and two worst solutions gave a ΔT of slightly over 10^{-7}. The outcomes are summarized in Table 1.

number of		*influential*	*relative difference* ΔT	
processors	*tasks*	*diversification*	*average*	*std. deviation*
2	100	no	$1.22\ 10^{-9}$	$1.05\ 10^{-9}$
2	100	yes	$9.73\ 10^{-10}$	$1.17\ 10^{-9}$
5	50	no	$5.85\ 10^{-6}$	$4.94\ 10^{-6}$
5	50	yes	$1.21\ 10^{-6}$	$5.60\ 10^{-7}$
10	100	no	$1.24\ 10^{-5}$	$1.72\ 10^{-6}$
10	100	yes	$1.84\ 10^{-6}$	$6.44\ 10^{-7}$
20	200	no	$1.10\ 10^{-5}$	$9.14\ 10^{-6}$
20	200	yes	$2.93\ 10^{-6}$	$1.33\ 10^{-6}$

Table 2: The results of the test runs. The same parameters were used for all runs. The search stopped after 50,000 non-improving moves.

The second test runs were to show the effect of the influential diversification component. Each problem was solved twice, once without and once with the influential diversification component. For each problem size, the results are averaged over ten different problems (of the same size). The outcomes are summarized in Table 2. The influential diversification indeed improves the quality of the solutions (yielding better outcomes for about 80% of the problems). Even more interesting, it produces a much smaller standard deviation in all cases except for the 2-processor case. This makes it possible to predict the quality of a solution more accurately. Improvements were less pronounced for problems with a small number m of processors.

For an additional test, we ran our procedure on a 10 bin/100 item Minmax bin packing problem, originally given by Graham (1984). This problem has an ideal weight of $49,999,998,902$ for minimizing the maximum weight received by any bin (though no one knows whether this weight is attainable). The best previously known result for this problem was obtained by an extended simulated annealing approach (Kämpke, 1988), where each random move is followed by a smoothing step, resulting in a weight of $50,002,124,855$. (This solution was found on only one trial from a large number of solution attempts applying different random generating seeds.)[5]

Because the scheduling problem described in this paper is isomorphic to the Minmax bin packing problem, our approach was directly applicable. The latest task finishing time T_i of a processor i is simply interpreted as the weight of bin i. Applying the tabu search approach without the influential diversification component we improved the best previous result by three additional digits, obtaining a weight of $50,000,0001,748$. We ran the procedure using the randomized best-fit heuristic to slightly modify the initial starting position. On more than 80% of these runs, the solutions were in

[5]Interestingly, Kämpke found that neither standard simulated annealing nor his improved extension could solve a small but "hard" two bin problem involving seven items of weight 5 and five items of weight 7. No matter how long the simulated annealing approaches were run, an optimal solution could not be found, except by a rare lucky choice of a random number seed. By contrast, both versions of our tabu search approach always found an optimal solution within at most 5 iterations after reaching the first local optimum.

the range between $50,000,010,000$ and $50,000,060,000$ (significantly better than the previous best found by Kämpke). The inclusion of influential diversification improved our results still further. The best solution was improved to $50,000,000,675$ and most solutions ($> 80\%$) fell in the range between $50,000,004,000$ and $50,000,022,000$. We note that the influential diversification step allowed the procedure to search longer, not faster. Without this component the search procedure can be stopped after $20,000$ moves fail to improve the current best solution, because it is unlikely after this point that a better solution will be found. This does not apply to the version with influential diversification where often an improvement can be observed after executing more than $50,000$ moves that do not improve the current best. (For this problem, it takes about 1 second to make 1000 moves on a MIPS computer.)

In summary, the short term memory component of the tabu search procedure makes it possible to reach good solutions quickly and the influential diversification strategy provides an opportunity for continued improvement over a significantly extended horizon.

Acknowledgment

We are indebted to Manuel Laguna for comments that have improved the quality of this paper.

This research was supported in part under the Air Force Office of Scientific Research and Office of Naval Research Contract F49620-90-C-0033 at the University of Colorado.

References

(de Werra and Hertz, 1989) D. de Werra and A. Hertz. Tabu search techniques: A tutorial and an application to neural networks. *OR Spectrum*, pages 131–141, 1989.

(Glover and Laguna, 1992) Fred Glover and Manuel Laguna. Tabu search. To appear in Colin Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Publishing, 1992.

(Glover and McMillan, 1986) Fred Glover and Claude McMillan. The general employee scheduling problem: An intergration of MS and AI. *Comput. & Ops. Res.*, 13(5):563–573, 1986.

(Graham, 1984) R. L. Graham. Combinatorial scheduling theory. In L. A. Steen, editor, *Mathematics Today*. Springer, New York, 1984. 3rd printing.

(Johnson *et al.*, 1991) David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation, part II (graph coloring and number partitioning). *Operations Research*, 1991.

(Kämpke, 1988) Thomas Kämpke. Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, 16:327–332, 1988.

(Skorin-Kapov, 1990) Jadranka Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

(Weber and Liebling, 1986) M. Weber and Th. M. Liebling. Euclidean matching problems and the metropolis algorithm. *ZOR Ser. A 30*, pages 85–110, 1986.