Journal of the Operational Research Society (2004) 55, 780–784

www.palgrave-journals.com/jors

A priority list based heuristic for the job shop problem: part 2 tabu search

Paul ME Shutler*

Nanyang Technological University, Singapore

Following a recent paper by the same author, a priority list-based tabu search heuristic is compared with the leading schedule-based tabu search heuristic of Nowicki and Smutnicki. More search neighbourhoods are required to achieve a given average makespan, but each priority list neighbourhood is searched much faster than the corresponding neighbourhood in the space of feasible schedules. Priority list-based tabu search therefore outperforms schedule-based tabu search in terms of elapsed CPU time.

Journal of the Operational Research Society (2004) **55,** 780–784. doi:10.1057/palgrave.jors.2601757
Published online 14 April 2004

Keywords: scheduling; heuristics; job shop problem; tabu search; priority lists

Introduction

In a recent paper[1] it was shown that priority list based stimulated annealing outperformed the leading schedule-based simulated annealing heuristic of van Laarhoven *et al.* [2] This paper extends the earlier work by showing that priority list based tabu search outperforms the leading schedule-based tabu search heuristic of Nowicki and Smutnicki.[3] For brevity, the reader is referred to the earlier paper for complete definitions, properties and examples of priority lists, a comprehensive analysis of the performance of the various heuristics on different computers, and a full discussion of the practical implications of this work.

The makespan of the priority list can be computed about four times faster than the makespan of the corresponding schedule,[1] but only at the cost of losing almost all of the information contained in the schedule, such as the start and end times of the operations. Without this information it is impossible to construct the critical path, so the family of search neighbourhoods based on the neighbourhood $\mathcal{N}_1$ of van Laarhoven *et al*[2] is also unavailable. Consequently, in the earlier paper, the simplest possible neighbourhood $\mathcal{N}_0$ obtained by swapping any pair of elements in a priority list was used. Since this neighbourhood is too large to be searched exhaustively, and since the only information known about its elements are their makespans, Monte Carlo-simulated annealing seemed to be the natural choice. In the event it performed remarkably well.

When computing the makespan of a priority list, however, complete information about the schedule is available should one choose to record it.[1,4] This suggests operating the priority list makespan algorithm in two versions: a fast version which computes the makespan alone, and a slow version which computes the full schedule. By applying the slow version to the current priority list it is possible to obtain the critical path, from which one can construct a priority list neighbourhood which mimics the schedule neighbourhood $\mathcal{N}_{1b}$ used by Nowicki and Smutnicki, and hence perform priority list-based tabu search. By applying the fast version to evaluate the makespans of the priority lists in this neighbourhood it is possible to retain most of the computational speed advantage of priority lists.

In the earlier paper it was shown that this fast version computes the makespan of a priority list about four times faster than Bellman's algorithm[5,6] computes the makespan of the corresponding schedule. For the problem sets studied here, the neighbourhood $\mathcal{N}_{1b}$ generally has around 10 or more elements, so the fast version of the code generally executes at least 10 times more frequently than the slow version. We would therefore expect, neighbourhood for neighbourhood, priority list based tabu search to operate slightly less than four times faster than schedule-based tabu search. Later, we shall see that this turns out to be the case.

In the earlier paper, it was shown that about 10% of the time taken to run the makespan algorithm could be saved by recording the job and machine ready times at some 'midpoint' part way through the priority list: if the next priority list differs from the current list only after the midpoint, the makespan algorithm need only be run from the midpoint on, rather than from the start of the list. This approach now takes on a greater significance since the elements of the priority list version of $\mathcal{N}_{1b}$ are naturally constructed in priority list order. So all the makespans in the

*Correspondence: PME Shutler, Mathematics and Mathematics Education AG, National Institute of Education, Nanyang Technological University, 1 Nanyang Walk, Singapore, 637616, Singapore.
E-mail: shutler@nie.edu.sg

neighbourhood can be computed in common up to the first pair swap, then all but that element of the neighbourhood up to the next pair swap and so on. The pattern of computation resembles a triangle rather than a rectangle, and so might be expected to reduce by half the time taken to evaluate all makespans in a neighbourhood.

The remainder of this paper consists of two main sections. In the first section, the priority list version of the neighbourhood $\mathcal{N}_{1b}$ is constructed and used to implement a tabu search heuristic which is rather simpler than the heuristic used by Nowicki and Smutnicki. It is also shown how the fact that $\mathcal{N}_{1b}$ is in priority list order can be used to compute the makespans of all the priority lists in the 'triangular' fashion mentioned above. In the second section, the effectiveness of the priority list-based tabu search heuristic is demonstrated by comparing its performance with that of the schedule-based tabu search heuristic of Nowicki and Smutnicki when applied to the three largest problem sets used in the earlier paper. Priority list-based tabu search performs significantly better on all three sets of problems.

Tabu search

Having computed the makespan of a priority list, the critical path of the corresponding schedule can be constructed by stepping back along the priority list until an entry `K` whose end time equals the makespan of the schedule. More generally, given an entry `K` on the critical path step back until an entry `KK` which ends at the same time that `K` starts and which is either part of the same job or executed on the same machine as `K`. Note that in the case of multiple critical paths the above approach effectively makes a random choice. For example, given the choice between an entry which corresponds to an operation on the same machine or an entry which corresponds to an operation of the same job, the entry which happens to come later in the priority list is selected, and it could be either.

The neighbourhood $\mathcal{N}_0$ used in the earlier paper was obtained by swapping any two entries in the priority list. Divide the entries of the priority which lie on the critical path into *blocks* of operations adjacent on the same machine in the normal way[3] and define the priority list version $\bar{\mathcal{N}}_{1b}$ to be the subset of $\mathcal{N}_0$ which corresponds to $\mathcal{N}_{1b}$. In other words, the neighbourhood $\bar{\mathcal{N}}_{1b}$ is obtained by swapping any pair of entries in the priority list which are either the first two or last two entries in a critical path block but excluding the first pair in the first block and the last pair in the last block. Note that $\bar{\mathcal{N}}_{1b}$ and $\mathcal{N}_{1b}$ in general *do not* coincide because swapping two entries in a priority list can result in a change in the schedule which is more complicated than simply swapping the corresponding operations.[1]

The priority list-based version of tabu search used in this paper is as follows. If a swap `K`↔ `KK` in $\bar{\mathcal{N}}_{1b}$ is made, identify the jobs `J` and `JJ` and the machine `M`. Place this information, but with `J` and `JJ` in *reverse order* on a pushdown stack, the tabu list, of maximum length `TABULENGTH`, the idea being to prevent an immediate return to the previous priority list. The next priority list is selected to have the least makespan among all the lists in $\bar{\mathcal{N}}_{1b}$ which are either not tabu or which improve upon the best makespan seen so far. Should the whole neighbourhood be tabu, simply clear the tabu list. Should this next priority list not improve on the current best makespan seen so far, a counter `STRIKE` is incremented by one, but if the new makespan is better `STRIKE` is reset to zero. Should `STRIKE` exceed a fixed `STRIKELIMIT` then simply begin the tabu search again from a randomly chosen priority list.

The elements of $\bar{\mathcal{N}}_{1b}$ are naturally constructed in priority list order, so it is possible to adapt the 'midpoint' trick in the earlier paper as follows. Execute the fast version of the makespan algorithm from the first entry `K`$=1$ up to the entry immediately preceding the first swap `K`↔ `KK` in $\bar{\mathcal{N}}_{1b}$. Temporarily record the job and machine ready times which prevailed at that point, make the swap, then execute until the end of the list. Return to that point again, restore the ready times, and execute from there up to just before the next swap in $\bar{\mathcal{N}}_{1b}$, and so on. The resulting 'triangular' pattern of computation might be expected to reduce the time taken to compute all the makespans in the neighbourhood by approximately a factor of 2. The time required to store then retrieve the ready times spoils this, however, but by counting the number of additional lines of code which have to be executed we find that the actual improvement is more realistically a factor of $\frac{10}{7}$. This factor will later be confirmed by the computational results.

Computational results

The performance of priority list-based tabu search was compared with the tabu search heuristic of Nowicki and Smutnicki on the five problems I_1–I_5 of size 15(machines) × 15(jobs) generated by Lawrence,[7] for which complete optimal solutions are known,[3,8] then on two sets of problems generated by Taillard[9] namely the set TA_{21}–TA_{30} of size 20×20, since it consists of problems which are known to be hard, and the set TA_{61}–TA_{70} of size 20×50, since it is the largest set of problems tackled systematically by Nowicki and Smutnicki. For TA_{21}–TA_{30} the reference points are the best makespans given by Taillard[10] and for TA_{61}–TA_{70} the best makespans obtained by Nowicki and Smutnicki, which are mostly proven optimal solutions.[3] Explicit values of these 'optimal' makespans for all the problems are given in column 4 of Table 1, and those which are not proven are indicated by an asterisk.

All the results presented in this paper were computed on a Digital Alpha Sever 8400 5/440, so the problem of comparing results from different computers, discussed in the earlier paper,[1] does not arise. For convenience the data

Table 1 Parameters used to generate the data shown in Figures 1–6

				Tabu search on priority lists (data ○)		*Tabu search on schedules (data ●)*		
Problem set	*Mac's m*	*Jobs n*	*Optimal makespan*	TABU LENGTH	STRIKE LIMIT	maxt	maxl	*Backtracking parameters*
I_1–I_5	15	15	1268, 1397, 1196, 1233, 1222	8	2000	8	5	9600/1600
TA_{21}–TA_{30}	20	20	1644*, 1600*, 1557*, 1648*, 1596*, 1647*, 1680*, 1614*, 1625*, 1584*	12	10 000	8	5	10 000/6000
TA_{61}–TA_{70}	20	50	2868, 2902*, 2755, 2702, 2725, 2845, 2841*, 2784, 3071, 2995	110	10 000	8	5	10 000/6000

* = Not proven.

for the heuristic of Nowicki and Smutnicki which was already generated in the earlier paper[1] was reused. For full details as to how these data were generated, the reader is referred to than earlier paper, but the following is a minimal outline required to understand what the data represent.

Nowicki and Smutnicki employ a rather complicated backtracking procedure controlled by a pair of parameters. For full details of how these parameters are used the reader is referred to their original paper,[3] but the manner in which they operate is very similar to the variable STRIKELIMIT. Whenever their tabu search exceeds these backtracking parameters the current schedule is replaced by one seen earlier, taken from a pushdown stack. Since this stack has maximum length 5, their tabu search necessarily terminates after a finite number of neighbourhoods. Although this number can vary quite widely between different runs even on the same problem, on an average the number of neighbourhoods searched is roughly proportional to the size of the backtracking parameters.

The schedule-based data (●) in Figures 1–3 show the best makespans obtained *during a run* of the tabu search heuristic of Nowicki and Smutnicki, expressed as a function of the number of neighbourhoods searched. Each point represents the average makespan obtained during 10 parallel runs on each of the problems in the set, one point for each problem, for the backtracking parameters shown in Table 1, to allow direct comparison with the results reported by Nowicki and Smutnicki in their Table 5.[3] The priority list-based data (○) in Figures 1–3 show the corresponding results for the priority list-based tabu search heuristic, averaged over 100 runs on each problem and averaged over all the problems in the set, for the values of TABULENGTH and STRIKELIMIT indicated in Table 1.

As is clear from Figures 1–3, the priority list-based tabu search never terminates by itself, but can be halted at any desired number of neighbourhoods searched. Figures 4–6 therefore show the same average makespans as those shown in Figures 1–3, but this time as a function of average CPU time per run $\bar{t}$ instead of number of neighbourhoods. The heuristic of Nowicki and Smutnicki, on the other hand, does terminate by itself so, in order to introduce some variation,

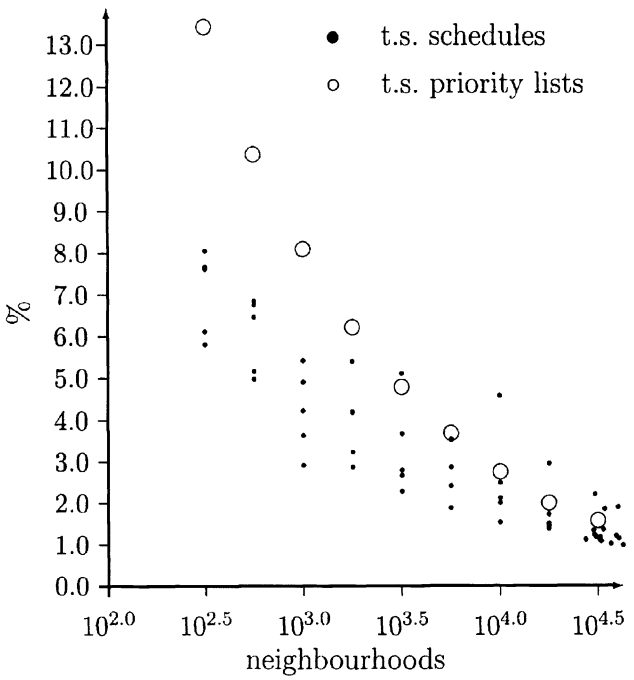

Figure 1 Percentage makespan above optimum % as a function of the number of neighbourhoods for both heuristics applied to problems I_1–I_5.

the schedule-based data (●) in Figures 4–6 are obtained by successively halving or doubling the backtracking parameters. That these data are equally spaced on a logarithmic scale confirms the earlier observation that the length of a run of their heuristic is on an average proportional to the size of the backtracking parameters. For the purposes of reference, the results obtained for the backtracking parameters shown in Table 1 are indicated by the larger-sized (●) symbols shown in Figures 4–6.

Conclusion

Figure 4–6 show that priority list-based tabu search heuristic does indeed outperform the schedule-based tabu search

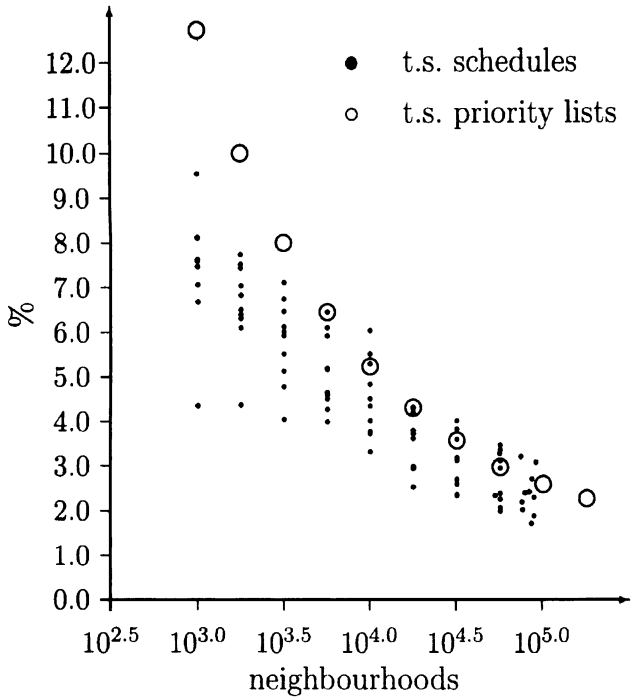

Figure 2 Percentage makespan above best-known value % as a function of the number of neighbourhoods for both heuristics applied to problems TA_{21}–TA_{30}.

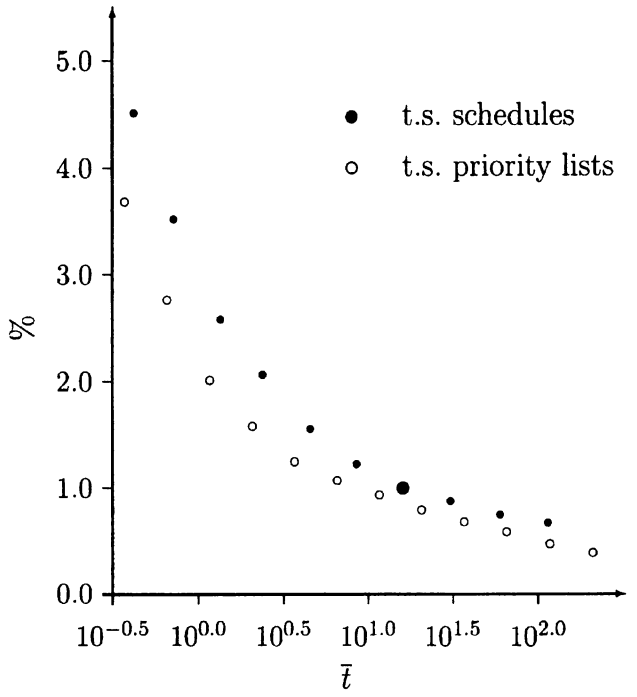

Figure 4 Average percentage makespan above optimum % as a function of average CPU time per run $\bar{t}$ for both heuristics applied to problems I_1–I_5.

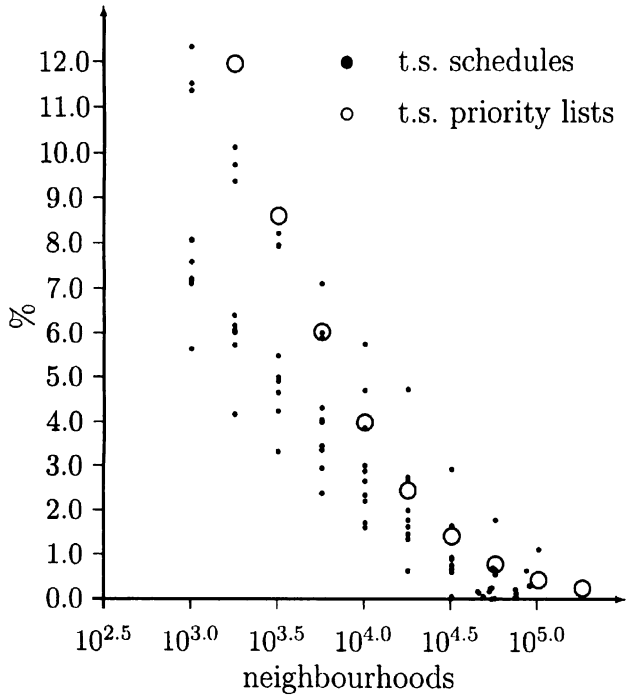

Figure 3 Percentage makespan above optimum % as a function of the number of neighbourhoods for both heuristics applied to problems TA_{61}–TA_{70}.

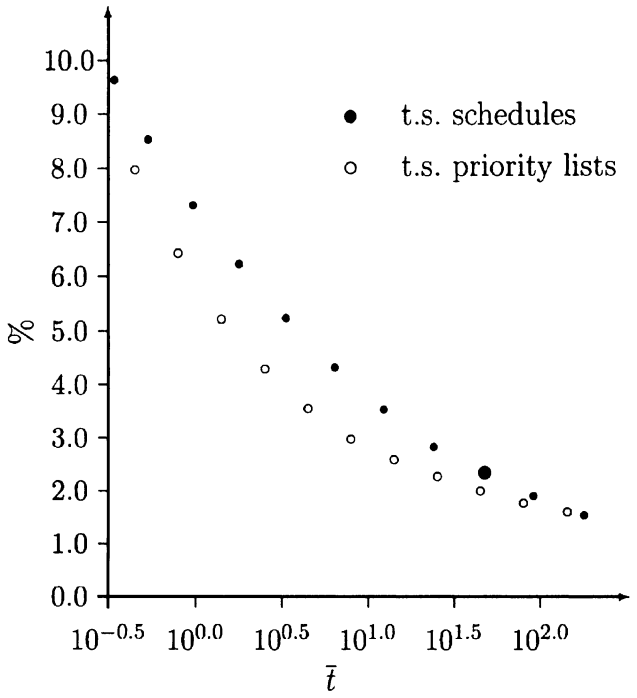

Figure 5 Average percentage makespan above best-known value % as a function of average CPU time per run $\bar{t}$ for both heuristics applied to problems TA_{21}–TA_{30}.

heuristic of Nowicki and Smutnicki, if performance is measured as average makespan obtained *versus* elapsed CPU time. For example, in the middle of the range of Figure 6 the priority list data (○) are horizontally displaced about 0.5 logarithm units to the left of the schedule to the data (•), showing that priority list-based tabu search is

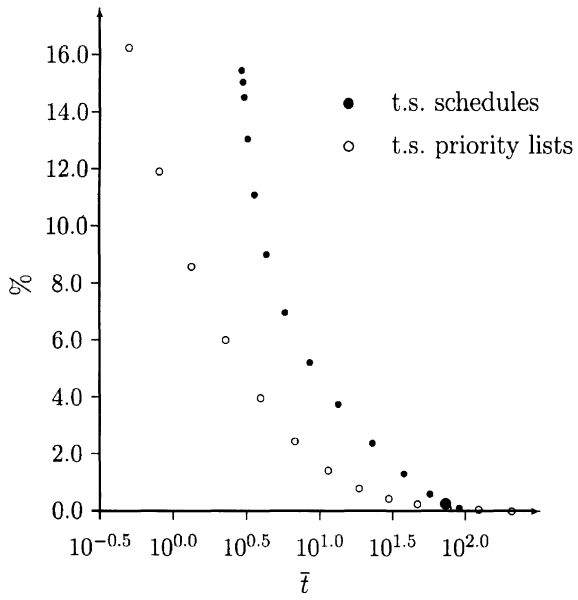

Figure 6 Average percentage makespan above optimum % as a function of average CPU time per run $\bar{t}$ for both heuristics applied to problems TA_{61}–TA_{70}.

about a factor $10^{0.5}$ faster than schedule-based tabu search. For Figure 4 it is about 0.25 logarithm units, and somewhere between the two for Figure 5.

Given that the priority list neighbourhood $\bar{\mathcal{N}}_{1b}$ is an imperfect approximation to the schedule neighbourhood $\mathcal{N}_{1b}$, however, schedule-based tabu search would have been expected to outperform priority list-based tabu search if performance is measured as makespan *versus* number of neighbourhoods searched. This is indeed the case, and in the middle of the range of Figure 3 the priority list data (○) are horizontally displaced about 0.25 logarithm units to the right of the schedule data (•), showing that priority list-based tabu search is about a factor $10^{0.25}$ less efficient in neighbourhood terms than schedule-based tabu search. For Figure 1 it is about 0.5 logarithm units and somewhere between the two for Figure 2.

Putting these two observations together confirms the original estimates. It was expected that priority list-based tabu search would execute a little less than four times faster than schedule-based tabu search due to the efficiency of the makespan code compared to Bellman's algorithm. It was also expected to execute about $\frac{10}{7}$ times faster due to the 'triangular' computation of makespans in a neighbourhood. Taken together the overall speed up should be a little less than $4\times\frac{10}{7} = 5.7$ which is very close to the relative shift of about $10^{0.5} \times 10^{0.25} = 5.6$ observed between the per neighbourhood and per CPU time graphs.

As well as outperforming schedule-based tabu search, priority list-based tabu search is simpler and easier to program. The `FORTRAN` inplementation of the former ran to 860 lines of code, whereas the latter occupied 430 lines, exactly half the length. In the case of TA_{62} and TA_{67} priority list-based tabu search produced shorter makespans than the 'optimal' 2902 and 2841 recorded in Table 1, the best seen being 2882 and 2825, respectively.

References

1 Shutler PME (2003). A priority list based heuristic for the job shop problem. *J Opl Res Soc* **54**: 571–584.
2 van Laarhoven PJM, Aarts EHL and Lenstra JK (1992). Job shop scheduling by simulated annealing. *Opns Res* **40**: 113–125.
3 Nowicki E and Smutnicki C (1996). A fast taboo search algorithm for the job shop problem. *Mgmt Sci* **42**: 797–813.
4 Golenko-Ginzburg D and Sims JA (1992). Using permutation spaces in job-shop scheduling. *Asia Pacific J Opl Res* **9**: 183–193.
5 Bellman RE (1958). On a routing problem. *Quart Appl Math* **16**: 87–90.
6 Taillard ED (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA J Comput* **6**: 108–117.
7 Lawrence S (1984). Resource constrained scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.
8 Vaessens RJM, Aarts EHL and Lenstra JK (1996). Job shop scheduling by local search. *INFORMS J Comput* **8**: 302–317.
9 Taillard E (1993). Benchmarks for basic scheduling problems. *Eur J Opl Res* **64**: 278–285.
10 Taillard E (2002). http://www.eivd.ch/ina/collaborateurs/etd/default.htm.

Received July 2003;
accepted February 2004 after one revision