A pruning pattern list approach to the permutation flowshop scheduling problem

Takeshi Yamada
NTT Communication Science Laboratories,
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, JAPAN
E-mail : yamada@cslab.kecl.ntt.co.jp

Abstract

This paper investigates an approach to the permutation flowshop scheduling problem based on Tabu Search with an additional memory structure called a 'pruning pattern list'. The pruning pattern list approach allows a better use of the critical block information. A solution of the flowshop scheduling problem is represented by a permutation of job numbers. A pruning pattern is generated from a solution by replacing job numbers inside a critical block with 'wild cards' so that solutions that 'match' the pattern would be excluded from the search. A set of pruning patterns, which is called a 'pruning pattern list', is used to navigate the search by avoiding solutions that would match any pattern on the list. Computational experiments using benchmark problems demonstrate the effectiveness of the proposed approach.

1 Introduction

The scheduling problem investigated in this paper is called the permutation flowshop scheduling problem (PFSP) and is conventionally designated as $n/m/P/C_{max}$[1], where n jobs have to be processed on m machines in the same order. The processing of each job on each machine is an operation, which requires the exclusive use of the machine for an uninterrupted duration called the processing time. P indicates that only the permutation schedules are considered, where the order in which each machine processes the jobs is identical for all machines. Hence, a schedule is uniquely represented by a permutation of jobs. The objective here is to find a schedule that minimizes the makespan C_{max}, the time at which the last job is completed on the last machine. The problem is strongly $\mathcal{NP}$-hard[1], therefore complete enumeration methods are not computationally practical as the problem size increases. Approximation methods to solve the problem are categorized to two types: one is constructive methods and another is iterative improvement methods. Constructive methods have been proposed by Campbell et al.[2], Dannenbring[3] and Nawaz et al.[4]. As for iterative improvement methods, Osman and Potts[8] and Ogbu and Smith[6] proposed simulated annealing methods, and Widmer and Herz[7] and Taillard[10] proposed tabu search methods. More recently, Nowicki and Smutnicki proposed a very powerful approximation method based on tabu search[5].

In this paper, an approximation method based on Tabu Search with an additional memory structure called "pruning pattern list" is proposed. In short, the pruning pattern list approach allows a better use of the critical block information. A pruning pattern is constructed from a solution represented by a permutation of job numbers by replacing some of job numbers inside a critical block by wild cards ($\star$-s) so that solutions that 'match' the pattern will be excluded from the search hereafter. This means that any modification of a solution, regardless of inside or outside the critical block, is discouraged if the resulting sequence would match the pattern. A list of pruning patterns generated from good schedules collected in the course of a search process is used to prevent the search from visiting already searched and no longer interesting region again and again.

2 Basic concepts

A critical path is a sequence of operations starting from the first operation on the first machine M_1 and ending with the last operation on the last machine M_m. The starting time of each operation on the path, except for the first one, is equal to the completion time of its preceding operation—that is, there is no idle time along the path. Thus, the length of the critical path is the sum of the processing times of all the operations on the path and equals to C_{max}. There can be more than one critical path in a schedule.

The operations on a critical path can be partitioned into subsequences, called *critical blocks*, according to their associated machines. A critical block consists of maximal consecutive operations on the same machine, or to put it more simply, a subsequence of associated jobs. Most of the following notations have previously been used by Nowicki and Smutnicki[5]. Consider a schedule represented by a permutation π. Let $B_1, \dots B_k$ be a set of all critical blocks that contains more than one job and let m_l be the index of the machine associated with B_l. Let $\pi(u_l)$ be the first job of B_l (and the last job of B_{l-1}). Then the 'inside' of B_l, denoted by $\hat{B}_l$, is defined as follows:

$$\hat{B}_l = \begin{cases} B_l \setminus \{\pi(u_{l+1})\} & \text{if } l = 1 \text{ and } m_l = 1 \\ B_l \setminus \{\pi(u_l)\} & \text{if } l = k \text{ and } m_l = n \\ B_l \setminus \{\pi(u_l), \pi(u_{l+1})\} & \text{otherwise.} \end{cases}$$

Figure 1 shows an example of schedule $\pi = 4, 5, 6, 1, 2, 3, 8, 7$ for a problem with $n = 8$ jobs and $m = 6$ machines represented by a grid graph, that is taken from [5]. In the figure, the vertical axis corresponds to machines and the horizontal axis to jobs. Each circle represents an operation, and arrows precedence relation between operations. A critical path is marked by thin lines. In this example, there are four critical blocks B_1, B_2, B_3, B_4 that contain more than one job. B_2 on machine 3, for example, consists of four jobs $5, 6, 1$ and 2, and $\hat{B}_2$ consists of jobs 6 and 1. Likewise $\hat{B}_4$ on machine 6 consists of jobs 8 and 7.

A neighborhood $N(x)$ of a point x in a search space can be defined as a set of new points that can be reached from x by exactly one transition or move (a single perturbation of x). One of the well-known transition operators for PFSP is the *shift move* which takes a job from its current position and re-inserts it in another position. Let $v = (a, b)$ be a pair of positions in π. Here, v defines a move that removes the job $\pi(a)$ from a position a and re-inserts it in a position b. If $a < b$, the resulting schedule is represented by $\pi_v = \pi(1), \dots, \pi(a-1)\pi(a+1), \dots, \pi(b), \pi(a), \pi(b+1), \dots, \pi(n)$, and if $a > b$, $\pi_v = \pi(1), \dots, \pi(b), \pi(a), \pi(b+1), \dots, \pi(a-1)\pi(a+1), \dots, \pi(n)$. A neighborhood $N(V, \pi)$ is defined as the set of all schedules obtained by shift moves in $V = \{(a, b) : b \notin \{a-1, a\}, a, b \in \{1, \dots, n\}\}$.

Let $W_l(\pi)$ be a set of moves restricted to the inside of B_l, namely $W_l(\pi) = \{(a, b) \in V | a, b \in \hat{B}_l\}$ and $W(\pi) = \bigcup_{l=1}^{k} W_l(\pi)$, then the so-called 'block property' is formulated as follows:

Property A (block property): For any schedule β, $\beta \in \mathcal{N}(W(\pi), \pi) \Longrightarrow C_{max}(\beta) \geq C_{max}(\pi)$.

According to property A above, no move in $W(\pi)$ can directly improve schedule π. Therefore, it is reasonable, for computational efficiency, to reduce the size of the neighborhood $N(V, \pi)$ by eliminating moves in $W(\pi)$, and to use a new neighborhood $N(V \setminus W(\pi), \pi)$, which we call here a "critical block neighborhood".

3 Tabu search

Nowicki and Smutnicki [5] proposed a Tabu Search (TS) method for the PFSP. The tabu list used in their approach is described as follows: when $v = (a, b)$ is performed on π, a pair of jobs $(\pi(a), \pi(a+1))$ (if $a < b$) or $(\pi(a-1), \pi(a))$ (if $a > b$) is stored as 'tabu' in the tabu list T of length tl. Then, a move $v = (a, b)$ cannot be performed on β if $a < b$ and $\{(\beta(j), \beta(a)) \mid j = a+1, \dots, b\} \cap T \neq \phi$ or if $a > b$ and $\{(\beta(a), \beta(j)) \mid j = b, \dots, a-1\} \cap T \neq \phi$. The neighborhood

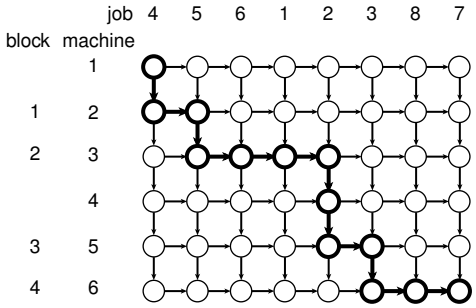

Figure 1: A grid graph representation of a solution to a problem of 8 jobs and 6 machines.

$N(V \setminus W(\pi), \pi)$, which is a subset of $N(V, \pi)$, is still too big for problems of even moderate size. Thus, it is impracticable to evaluate all the neighbors in $N(V \setminus W(\pi), \pi)$, so Nowicki and Smutnicki use a subset of $N(V \setminus W(\pi), \pi)$ as a neighborhood structure. Alternatively, one can make the evaluation probabilistic: a member α from $N(V \setminus W(\pi), \pi)$ is selected at random and accepted if $C_{max}(\alpha) < C_{max}(\pi)$, otherwise probabilistically accepted according to the Metropolis probability $P_c(\alpha) = \exp(-\Delta C_{max}/c)$, where $\Delta C_{max} = C_{max}(\alpha) - C_{max}(\pi)$.

4 Pruning pattern

A pruning pattern $[\pi]_l$ associated with π and its critical block B_l is derived from π by replacing the jobs that belong to $\hat{B}_l$ by $\star$ as follows:

$$[\pi]_l(j) = \begin{cases} \star & \text{if } j \in \hat{B}_l, \\ \pi(j) & \text{otherwise.} \end{cases}$$

For example, the pruning pattern corresponding to $\pi = 4, 5, 6, 1, 2, 3, 8, 7$ and B_2 in Figure 1 is $[\pi]_2 = 4, 5, \star, \star, 2, 3, 8, 7$. The makespan of $[\pi]_l$ is defined as the makespan of π, namely, $C_{max}([\pi]_l) := C_{max}(\pi)$. The $\star$ means a wild card, and a permutation β 'matches' $[\pi]_l$ if $\beta(j) = [\pi]_l(j)$ at all but $\star$ positions. For example $4, 5, 6, 1, 2, 3, 8, 7$ and $4, 5, 1, 6, 2, 3, 8, 7$ both match $[\pi]_2$. $[\pi]_l$ and a set of all permutations that match $[\pi]_l$ are identified. It is clear that $\mathcal{N}(W_l(\pi), \pi) \subset [\pi]_l$. The block property can be reformulated using $[\pi]_l$ as follows:

Property B: For any schedule β and any pruning pattern $[\pi]_l$, $\beta \in [\pi]_l \Longrightarrow C_{max}(\beta) \geq C_{max}([\pi]_l)$.

Property C below is a corollary of Property B:

Property C: For any two pruning patterns $[\pi]_i$ and $[\phi]_j$, $[\pi]_i \subset [\phi]_j \Longrightarrow C_{max}([\pi]_i) \geq C_{max}([\phi]_j)$.

Property B suggests that when a new, possibly good, solution π is found during the search, $[\pi]_l$ identifies a region where no solutions are better than π, and that excluding $[\pi]_l$ from the search space can reduce the size of the search space without eliminating the global optima.

5 Pruning pattern list approach

The basic idea of the pruning pattern list approach is to reduce the size of the search space effectively through storing the 'important' pruning patterns that correspond to a long critical block of a good solution. Let $[\pi]$ (without suffix) be the pruning pattern that corresponds to the longest critical block of schedule π. The pruning pattern list PL with length pl is maintained and updated as shown in Figure 2. In the figure, $N(\pi)$ represents the neighborhood of π. $N(V \setminus W(\pi), \pi)$ or its subset is normally used as $N(\pi)$. There is no good reason to keep patterns that are rarely accessed in the list. Therefore such patterns are replaced by new patterns. According to property C, it is

1. At the beginning of the search, PL is initialized as a list of empty pl elements.
2. After $N(\pi)$ is calculated from π, $N^P(\pi)$ is initialized as $N(\pi)$. For each $\alpha \in N^P(\pi)$, if α matches any pattern $[\beta] \in PL$, then α is removed from $N^P(\pi)$, resulting $N^P(\pi) := N^P(\pi) \setminus \alpha$, and the 'access count' of $[\beta]$ is incremented. $N^P(\pi)$ is used as the new neighborhood of π.
3. When a cost-decreasing solution α is selected from the neighborhood $N^P(\pi)$ of the current solution π ($C_{max}(\alpha) < C_{max}(\pi)$), then its longest pruning pattern $[\alpha]$ is stored in the list.
4. All the existing patterns $[\beta]$ in the list that are subsets of a newly stored pattern $[\alpha]$ ($[\beta] \subset [\alpha]$) are removed and substituted by empty patterns.
5. If the number of non-empty patterns on PL exceeds pl, then the least accessed pattern $[\gamma]$ is removed from the list.

Figure 2: The pruning pattern list approach to the PFSP.

also not necessary to keep pruning patterns in the list that are subsets of other patterns. Thus they are removed.

Starting from an initial solution, the local search iteratively replaces the current solution with one of its neighbors. In advanced local search strategies, such as Simulated Annealing (SA) and TS, cost-increasing neighbors can be accepted as well as cost-decreasing ones. Accepting cost-increasing moves enables the search to escape from the local optima, but this may also cause revisiting of previously evaluated points in the search space – something that is wasteful of computing resources in itself, and which also means the search is not adequately diversified. One of the main aims of tabu search is to discourage such revisiting. This can be accomplished by means of an explicit 'tabu list' of points previously visited, but normally it is easier, and more efficient, to record specific attributes of such points, or of moves that would lead towards them. Nevertheless, recording attributes of the points, and not the points themselves, can risk treating even moves that are better than any solution obtained so far as tabu. This is one of the main reasons why aspiration criteria should be introduced into TS.

In the case of the pruning pattern list approach, property B guarantees that a move that matches a pruning pattern is never better than the best solution obtained so far. This means that an aspiration criterion is not necessary, and it also means that the pruning pattern list can serve as a longer-term memory to prevent the search getting stuck in an already searched and no longer interesting region. Unlike the tabu list, an old pattern can remain in the pruning list as long as it is accessed frequently. If $N(V \setminus W(\pi), \pi)$ or its subset is used as the neighborhood of the current solution π, it does not contain any solution that matches $[\pi]_l$, even without PL. However, it is still possible that the region $[\pi]_l$ would be revisited in some later stages without any better solutions than π being found. The size of PL is fixed to pl mainly for computational efficiency. Theoretically, $N^P(\pi)$ may not satisfy the connectivity property that $N(V \setminus W(\pi), \pi)$ does. It should be noted that the pruning pattern list is treated in a similar way that the population is treated in Genetic Algorithms (GAs), especially in the steady state model [12]. The access count of the pruning pattern corresponds to the fitness function of the individual in GAs.

6 Experimental results

The pruning pattern list approach described in the previous section can be embedded into any local search method, including TS, SA, and even GAs [13, 9]. Here a simple probabilistic version of the TS described in Section 3 is used as a test case. The programs are coded in C.

The graph on the left side of Figure 3 shows the time evolution of the makespan averaged over 30 runs of the TS with and without the pruning pattern list. This uses the ta041 problem, which is one of Taillard's benchmarks for size 50×20 (50 jobs and 10 machines) [11]. The parameters used are $c = 3.0, tl = 7$ and $pl = 10$. The pruning pattern list is updated after every 10000 TS evaluations for 700 times (thus total 10000×700 TS evaluations). One run takes about 10 minutes on an HP 700 workstation. The best and worst makespans obtained among 30 runs are 3000 and 3016 respectively when TS with the pruning pattern list is used, and 3010 and 3025 when TS without the pruning pattern list is used. It can be seen that the pruning pattern list approach improves both the solution quality and the speed. Without the pruning pattern list 12 runs out of 30 were trapped at a local minimum of makespan = 3025, which Nowicki and Smutnicki reported as the new reference makespan for this problem[5]. This suggests that their TS method can also be improved by incorporating the pruning pattern list approach.

The pruning pattern list approach is also applied to 50×20 (50 jobs and 20 machines) problems of Taillard's benchmarks (ta051 - ta060). The parameters used are the same as in the ta041 case and the results are averaged over 10 replications instead of 30. The results are summarized in Table 1, where the columns labeled TS+PL and TS show the average performance with and without the pruning pattern list respectively. The column labeled NOW shows the best makespan reported in[5]. The results of TS with the pruning pattern list generally outperform the results of TS without the pruning pattern list, except for the ta057 problem. As typical examples, the graph on the right side of Figure 3 shows the time evolution of the average makespan for ta051 problem. It also includes the computationally equivalent MSXF-GA results reported in [9] for comparison. Figure 4 shows the time evolution for other 50×20 problems.

prob	TS+PL	TS	NOW (best)
051	3859.4	3866.0	3875
052	3708.0	3713.4	3715
053	3653.9	3657.4	3668
054	3734.0	3737.2	3752
055	3617.1	3625.7	3635
056	3689.3	3691.7	3698
057	3712.8	3712.4	3716
058	3701.4	3706.8	3709
059	3756.9	3762.3	3765
060	3765.7	3767.3	3777

Table 1: Comparison with and without the pruning pattern list for problems size 50×20.

To see how the proposed approach behaves with growing problem sizes, the experiments are extended to Taillard's 100×20 (ta081 - ta090), 200×20 (ta101 - ta110) and 500×20 (ta111 - ta120) problems. For 100×20 (ta081 - ta090), 200×20 (ta101 - ta110) problems, the results are summarized in Table 2. Here, the pruning pattern list is updated after every 10000 TS evaluations for 160 times, which resulted in comparable and slightly better performances than those reported in [5]. The parameters used are are $c = 3.0, tl = 7$ and $pl = 5$. and the results are averaged over 10 replications. One run for each 100×20 and 200×20 problems takes about 6 and 12 minutes on an HP 700 workstation respectively. It is observed that the computational overhead of the pruning pattern calculation is about 8 % when $pl = 5$ and increases linearly as the size of PL. According to some preliminary experiments, the performance of the algorithm seems not sensitive to the length

of the pruning pattern list as long as $pl \geq 5$. It can be seen that the results of TS with the pruning pattern list generally outperform the results of TS without the pruning pattern list with some exceptions, whereas the gaps between TS+PL and TS become decreasing and less significant (less than 0.1 %).

prob	TS+PL	TS	NOW (best)	prob	TS+PL	TS	NOW (best)
081	6270.6	6270.2	6286	101	11285.5	11305.0	11294
082	6228.9	6233.0	6241	102	11358.5	11349.6	11420
083	6309.8	6314.8	6329	103	11428.3	11430.3	11446
084	6302.5	6306.6	6306	104	11357.7	11366.9	11347
085	6358.1	6362.1	6377	105	11301.5	11310.3	11311
086	6420.8	6431.1	6437	106	11269.7	11275.0	11282
087	6328.2	6333.6	6346	107	11463.8	11459.9	11456
088	6470.1	6471.5	6481	108	11413.5	11420.9	11415
089	6327.0	6332.2	6358	109	11302.8	11313.2	11343
090	6469.3	6476.9	6465	110	11404.6	11418.3	11422

Table 2: Comparison with and without the pruning pattern list for problems size 100×20 and 200×20.

Table 2 shows the results for Taillard's 500×20 (ta111 - ta120) problems. In the columns TS+PL and TS, the average, best and worst results over 10 replications are reported respectively. To keep the results comparable with those reported in [5], the number of iterations is doubled and the pruning pattern list is updated after every 10000 TS evaluations for 320 times with the other parameters same as 100×20 and 200×20 problems. Due to the increase both in the problem size and the number of iterations, one run now takes about 60 minuts. The same tendencies are observed here again: TS+PL runs are generally better than TS runs with some exceptions whereas the gaps between TS+PL and TS become decreasing.

Finally, to see the maximal performace of the TS+PL method, the number of iterations are doubled again for TS+PL runs only, resulting in best, worst and average makespans shown in the columns under the label "TS+PL (long)" in Table 2.

prob	TS+PL		TS		TS+PL (long)		NOW (best)
	average	[best worst]	average	[best worst]	average	[best worst]	
111	26324.0	[26198 26398]	26321.0	[26263 26403]	26159.6	[26139 26182]	26189
112	26687.6	[26633 26735]	26717.0	[26629 26805]	26625.5	[26600 26642]	26629
113	26465.6	[26435 26522]	26483.6	[26466 26512]	26431.4	[26417 26435]	26458
114	26547.2	[26522 26590]	26555.6	[26533 26584]	26525.0	[26520 26541]	26549
115	26371.4	[26354 26412]	26399.6	[26383 26413]	26356.2	[26354 26366]	26404
116	26589.2	[26535 26621]	26584.0	[26564 26621]	26516.3	[26498 26543]	26581
117	26432.4	[26412 26481]	26438.4	[26412 26463]	26413.0	[26412 26422]	26461
118	26678.8	[26619 26748]	26659.4	[26619 26736]	26615.9	[26615 26619]	26615
119	26149.8	[26097 26196]	26151.0	[26101 26235]	26078.8	[26050 26103]	26083
120	26600.0	[26546 26669]	26608.6	[26572 26684]	26526.4	[26497 26552]	26527

Table 3: Computational results for problems size 500×20.

7 Conclusion

The pruning pattern list approach to the makespan-minimizing permutation flowshop scheduling problem is proposed and embedded into a Tabu Search method. The experimental results

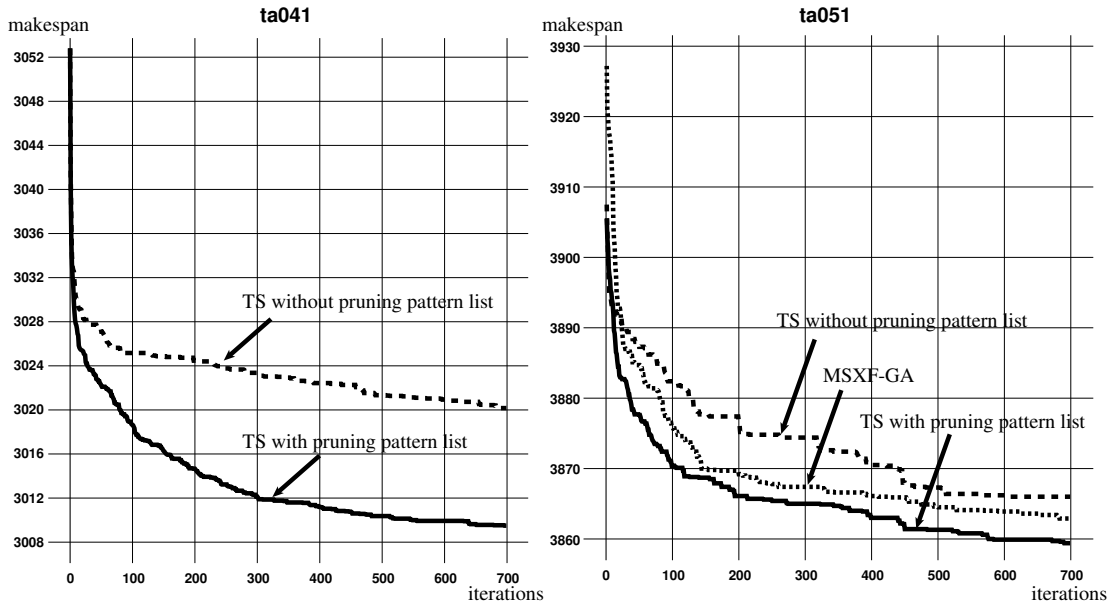

Figure 3: The time evolutions of makespans for the ta041 (50 jobs and 10 machines) and ta051 (50 jobs and 20 machines) problems

demonstrate the effectiveness of the proposed approach especially for medium-size problems. Good performances are also observed for large-size problems, however the gap between with and without the pruning pattern list approach seems to be decreasing, which would be left for future research. Future research will also aim to implement more efficient ways to scan the pruning pattern list and find a match more quickly. The proposed approach can also be embedded into GAs. However, the implementation details including whether each individual should have its own pruning pattern list and should occasionally exchange it with others, or a single list should be shared by all the members in the population are yet to be investigated.

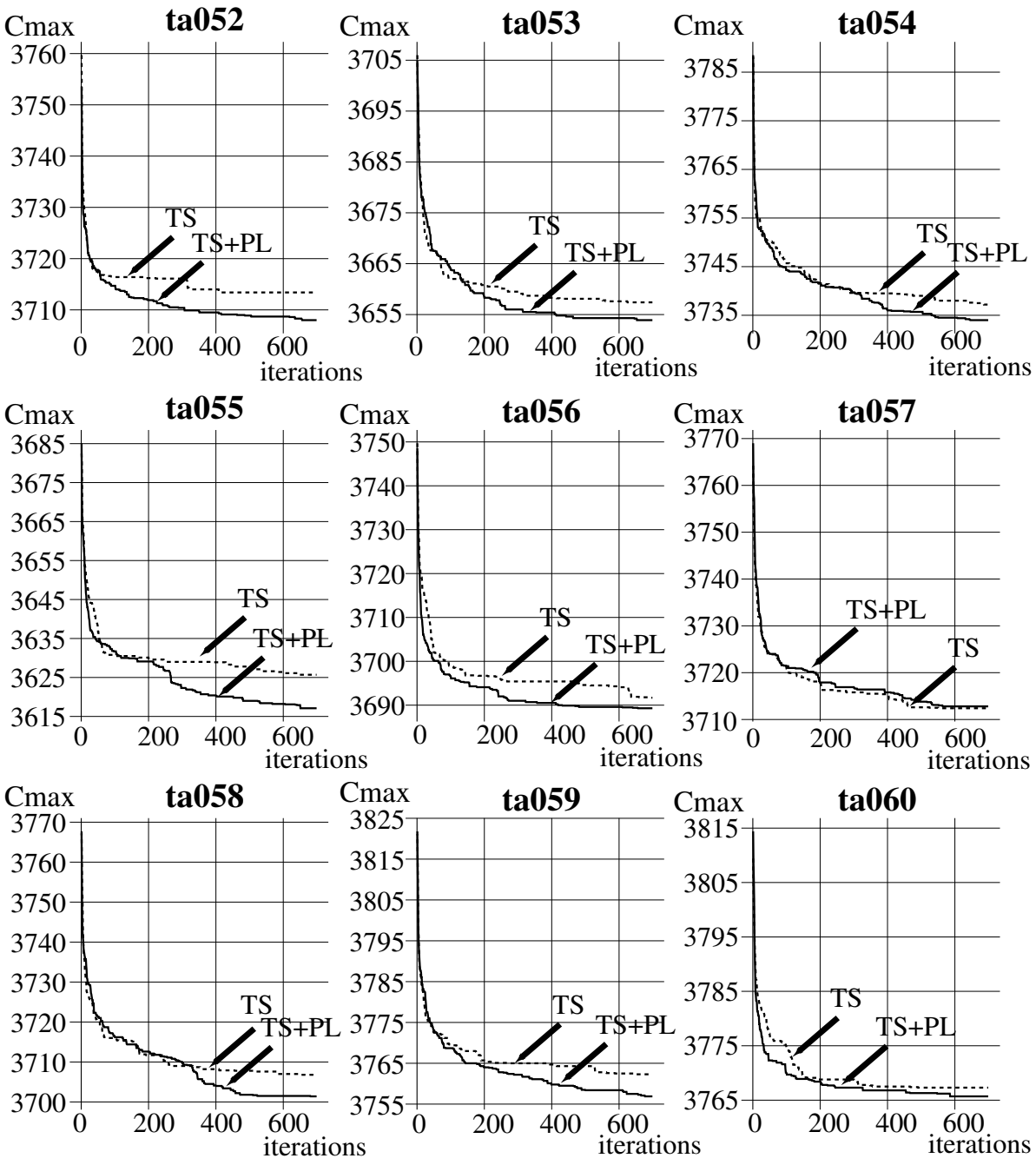

Figure 4: The time evolutions for the other nine Taillard problems of 50 jobs and 20 machines (ta052 – ta060)

References

[1] A.H.G. Rinnooy Kan. *Machine Scheduling Problems: Classification, Complexity and Computations.* Freeman, San Francisco, 1979.

[2] H.G. Campbell, R.A. Dudek and M.L. Smith. A heuristic algorithm for the n job m machine sequencing problem. *Management Science* 16, 630–637, 1970.

[3] D.G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11), 1174–1182, 1977.

[4] M. Nawaz, E.E. Enscore Jr. and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA*, 11(1), 91–95, 1983.

[5] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91, 160–175, 1996.

[6] F.A. Ogbu and D.K. Smith. Simulated annealing for the permutation flowshop problem. *OMEGA*, 19(1), 64–67, 1990.

[7] M. Widmer and A. Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41, 186–193, 1989.

[8] I.H. Osman and C.N. Potts. Simulated annealing for permutation flow-shop scheduling. *OMEGA*, 17(6), 551–557, 1989.

[9] C.R. Reeves and T. Yamada. Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation journal*, 6(1), 45–60, 1998.

[10] E. Taillard. Some efficient heuristic methods for flow shop sequencing. *European Journal of Operational Research*, 47, 65–74, 1990.

[11] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64, 278–285, 1993.

[12] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 133–140, 1989.

[13] T. Yamada and C.R. Reeves. Permutation flowshop scheduling by genetic local search. In *Proceedings of the 2nd IEE/IEEE Int. Conf. on Genetic ALgorithms in Engineering Systems (GALESIA '97)*, 232–238, 1997.