A Tabu Search for the Permutation Flow Shop Problem with Sequence Dependent Setup Times

Nicolau Santos, nicolau.santos@dcc.fc.up.pt

Faculdade de Ciências, Universidade do Porto; INESC Porto

João Pedro Pedroso, jpp@fc.up.pt

Faculdade de Ciências, Universidade do Porto; INESC Porto

Rui Rebelo, rrebelo@inescporto.pt

INESC Porto

Faculdade de Ciências, Universidade do Porto

Rua do Campo Alegre

4169-007 Porto, Portugal

Tel. (+351) 220 402 000

INESC Porto

Rua Dr. Roberto Frias, nº 378

4200 - 465 Porto, Portugal

Tel. (+351) 222 094 000

Abstract

In this work we present a metaheuristic method based on tabu search for solving the permutation flow shop scheduling problem with sequence dependent setup times, with the objective of minimizing total weighted tardiness. The problem is well known for its practical applications and for the difficulty in obtaining good solutions. The tabu search method proposed is based on the insertion neighborhood, and is characterized, at each iteration, by the selection and evaluation of a small subset of this neighborhood; this has consequences both on diversification and on speeding up the search. We also propose a speed-up based on book keeping information of the current solution, used for the evaluation of its neighbors.

Keywords: Combinatorial Optimization, Metaheuristics, Tabu Search, Flowshop, Scheduling

Introduction

Consider a flow shop with *n* jobs to process on *m* machines; let $p_{i,j}$ denote the known production time of job *j* in machine *i*. We assume that all jobs have the same sequence of operations on all the machines, so a solution is completely defined by a single permutation. This is known as the permutation flow shop problem, and is widely studied mainly for the objective of minimizing makespan, i.e., the completion time of the last job on the last machine. However, in order to model real-world production systems some additional details have to be taken into account. In our problem, each job *j* has a given due date d_j, and a weight w_j is used to assign priorities among the jobs. Unproductive times for reconfiguring a machine are sequence dependent; S_{ijk} represents the changeover time requiring to start the manufacture of a product *k* after a product *j* was produced on machine *i*. We denote as C_{ij} the completion time of job *j* in machine *i*. Given a permutation π, where π_j represents the job at position *j*, the completion times can be computed through the recursive formula

$$C_{i,\pi_j} = max\left\{C_{i,\pi_{j-1}} + S_{i,\pi_{j-1},\pi_j}; C_{i-1,\pi_j}\right\} + p_{i,\pi_j}$$

where $C_{0,\pi_j} = 0$, $S_{i,\pi_0,\pi_1} = 0$ and $C_{i,\pi_0} = 0$, for $i = 1,\ldots,m$, $j = 1,\ldots,n$. The tardiness of job *j* is $T_j = max\{C_{mj} - d_j, 0\}$, and the objective is to minimize the total weighted tardiness $W = \sum_{j=1}^{n} w_j T_j$.

This problem is NP-hard (Gupta, 1986), and the accelerator for incremental computation of the objective function proposed by Taillard (1990) is not applicable, as the computation of the objective function directly depends on the evaluation of the earliest completion time of each job on each machine; hence, large computational times are required for its solution. In the literature only a small number of authors focused on this problem; in (Parthasarathy & Rajendran, 1997a) and (Parthasarathy & Rajendran, 1997b) the authors propose a simulated annealing method, in (Rajendran & Ziegler, 2003) the authors develop a method to minimize the sum of weighted flowtime and weighted tardiness, in (Ruiz & Stützle, 2008) the authors apply an iterated greedy heuristics and present an extensive analysis of both proposed and adapted algorithms for this particular problem. A more generic review of setup related problems can be found in (Allahverdi, Ng, Cheng & Kovalyov, 2008).

In this article we present a tabu search heuristics, enhanced with a speedup method that allows significantly reducing the computational times. Follows an evaluation of the algorithm's performance, based on widely used literature benchmark tests. We end presenting some conclusions.

Tabu Search

Tabu Search was proposed by Glover (1986) as a method to guide a heuristic search through the solution space. Its main characteristic is the tabu list, a recent memory record that prevents the repetition of moves in the short run. In many cases, this avoids cycling, and leads the algorithm to explore promising regions. Our implementation is based on the insertion neighborhood, as suggested in (Taillard, 1990).

Initial solution

The starting solution for our algorithm is obtained with the heuristics proposed in (Ruiz & Stützle, 2008). This is an adaptation of the Nawaz, Enscore & Ham (1983) heuristic (NEH), and consists of two steps: first, the jobs are sorted by non-decreasing order of weighted due date (EWDD rule); then a greedy construction heuristics is used to build a new schedule (NEH_EWDD), according to the following steps:

1. remove a job from the begin of the EWDD solution;
2. tentatively insert this job in all possible positions of NEH_EWDD (initialy empty);

3. save the insertion that has the best objective.

The process is repeated until all jobs are scheduled; the final permutation is the NEH_EWDD solution.

Moves and neighborhood

Tabu search exploration is based on moving iteratively to a solution in the neighborhood. In our algorithm we use insertion moves: given a permutation π and a pair of positions $(i, j), i \neq j$, the permutation $\pi^{'}$ obtained by removing job at position *i* and inserting it at position *j* is:

$\pi^{'} = \pi_1, \dots, \pi_{i-1}, \pi_{i+1}, \dots, \pi_j, \pi_i, \pi_{j+1}, \dots, \pi_n$ if $i < j$;

$\pi^{'} = \pi_1, \dots, \pi_{j-1}, \pi_i, \pi_j, \dots, \pi_{i-1}, \pi_{i+1}, \dots, \pi_n$ if $j < i$.

Having a subset *U* of jobs, we define *N(U, π)* as the neighborhood of π, containing all the possible insertion moves of the jobs in *U*.

Tabu list and search strategy

In our implementation, tabu information is kept in an array: we assign a value τ_j to each job *j*, and at iteration *k* we say that job *j* is tabu (i.e., cannot be used in the current move) if $\tau_j > k$.

In a given iteration *k*, a job *j* chosen to perform the current move becomes tabu for *t* iterations, with $t_{min} \leq t \leq t_{max}$, and hence we set $\tau_j = k + t$. An exception is made when the best found solution is improved; in this case, we set $\tau_j = 0, \forall j$. (Both t_{min} and t_{max} are parameters of the algorithm.)

Evaluating the neighborhood generated by trying insertion among any pair of jobs is a demanding computational task, so we propose a neighborhood restriction: instead of evaluating the complete neighborhood we evaluate a set *R* of *r* randomly chosen jobs, with $r_{min} \leq r \leq r_{max}$ (r_{min} and r_{max} are parameters of the algorithm).

To illustrate the behavior of the algorithm, let us consider an example with seven jobs. Suppose that at a given iteration we have as incumbent solution (1, 2, 3, 4, 5, 6, 7). The operations to perform during a tabu search iteration are the following:

1. find the list of non-tabu moves *L*; suppose we obtain *L = {1,2,3,4}*;

2. draw *r* to find the number of jobs to evaluate; we obtain, e.g., *r = 2*;

3. randomly choose *r* jobs from *L*; we obtain, say, jobs 1 and 4, so *R = {1,4}*;

4. evaluate the solutions of *N(R, π)*; e.g., for job *1* the neighbor permutations are:

$$(\underline{2}, 1, 3, 4, 5, 6, 7)$$

$$(\underline{2, 3}, 1, 4, 5, 6, 7)$$

$$(\underline{2, 3, 4}, 1, 5, 6, 7)$$

$$(\underline{2, 3, 4, 5}, 1, 6, 7)$$

$$(\underline{2, 3, 4, 5, 6}, 1, 7)$$

$$(\underline{2, 3, 4, 5, 6, 7}, 1)$$

Then, the permutation that yields the best objective is chosen as the incumbent solution; suppose the first permutation is chosen; is this case, the job that will become tabu is *j=1*. The final step is to update the tabu list.

As can be seen from the previous example, when we evaluate the insertion of job 1, the information concerning the underlined partial permutations is not changed. Carefully bookkeeping this information across all the tentative insertions of each job leads to large computational savings.

The algorithm was experimentally calibrated by evenly dividing the used time in two phases. In the first phase, a highly diversified search is performed by setting $t_{min} = 1$ and $t_{max} = n$, forcing the method to choose with high probability different jobs to perform the move. In the second stage we set $t_{max} = 10$; this was observed to act as an intensification strategy, as the program conducts a more focused search. In both phases $r_{min} = 4$ and $r_{max} = 8$. Also a simple restart mechanism is implemented: if more than 300 iterations are performed without improving the best found solution we resume the incumbent solution to the best found so far and clear the tabu list, this process is also applied in phase transition. The parameters were set in order to obtain a good compromise between running time and solution quality.

Computational results

To evaluate the quality of the proposed algorithm we conducted a test with the four benchmark sets proposed in (Ruiz & Stützle, 2008). These are based on the 120 instances of Taillard (1993), which are extended with the inclusion of setup times, due dates, and weights. The original problems are organized in twelve groups of ten instances, containing different combinations of *n* jobs and *m* machines. The available combinations of n × m are: {20, 50, 100} × {5, 10, 20}, 200 × {10, 20} and 500 × 20. The processing times p are integers generated with uniform distribution in [1, 99]. The setup times are generated according to the processing times of the original instance, being at most 10, 50, 100 and 125% of the sum of the processing times in the original instances. Weights are integers, and are generated with uniform distribution in [1, 10]. Due dates are generated in a similar way to (Hasija & Rajendran, 2004). The process is summarized in three steps:

1. For each job j, calculate $P_j$the total processing time in all the m machines:

$$P_j = \sum_{i=1}^{m} p_{i,j}$$

2. Let S_j be the sum in all machines of the average setup time for all possible following jobs:

$$S_j = \sum_{i=1}^{m} \frac{\sum_{k=1,k\neq j}^{n} S_{i,j,k}}{n-1}$$

3. The due date d_j of job j is given by:

$$d_j = (P_j + S_j) \times (1 + u \times 3)$$

where u is a random number uniformly distributed in [0,1].

The four resulting groups are denominated ssd10, ssd50, ssd100 and ssd125, according to the ratio of setup times to processing times.

In order to have a term for comparison for our algorithm, we used the Iterated Greedy (IG) method proposed in (Ruiz & Stützle, 2008). IG is as adaptation of an algorithm initially proposed for makespan, which iteratively applies a greedy construction to an incumbent solution. At each iteration, four jobs are randomly removed, and inserted back by means of the greedy insertion heuristics of NEH_EWDD; the process is followed by a local optimization descent. Finally, an acceptance criterion similar to that of simulated annealing with constant temperature is used to update the incumbent solution.

In Table 1 we present the average of the relative percentage deviation errors (RPD), obtained by the algorithms IG, IGS and TS. Let *f* be the objective value obtained by a method on a given run, and *b* be the best known solution for the corresponding instance; *RPD* is defined as:

$$RPD = \frac{f-b}{b} \times 100.$$

IG represents the original results of (Ruiz & Stützle, 2008) with ten runs of each instance in an AMD Athlon XP 1600+ processor with 512 Mbytes of RAM and CPU time limit of n × m × 90 ms. IGS is our implementation of the iterated greedy algorithm with the referred speed up due to information bookkeeping, and TS is our proposed tabu search; results correspond to the average of 5 runs on each instance, with an allowed CPU time of n × m × 45 ms. In our experiment, tests were performed in a computer AMD Athlon 64 X2 Dual Core 3800+ with 2Gb of RAM. The algorithms are coded in mixed Python/Fortran wrapped with f2py (Peterson, 2009).

We can observe that, although using half of the computational time of IG, the results of IGS are far superior. The results of TS are still superior, obtaining the best results in 44 of the 48 groups of instances.

Conclusions

In this work we present a version of tabu search for the permutation flow shop problem, with sequence dependent setup times, and the objective of minimizing total weighted tardiness. The algorithm is based on the exploration of a small neighborhood with a dynamic tabu list, being supported by a speedup due to bookkeeping of solution information. Tests performed with standard benchmarks confirm the quality of the method proposed, as we obtain better results than those published using half of the CPU time. In future work we will apply the developed method to other flow shop problems, and experiment with different diversification/intensification strategies to try enhancing the performance even further.

Acknowledgements

This work was partially supported by the European Comission, Framework Programme 7 project FIT4U: Framework of Integrated Technologies for User Centred Products.

References

Allahverdi, A., Ng, C.T., Cheng, T.C.E. & Kovalyov, M.Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* , 985-1032.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* , 533-549.

Gupta, J. (1986). Flowshop schedules with sequence dependent setup times. *Journal of the Operations Research Society of Japan* , 206-219.

Hasija, S. & Rajendran, C. (2004). Scheduling in flowshops to minimize total tardiness of jobs. *International journal of production research* , 2289-2301.

Nawaz, M., Enscore Jr, E.E. & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* , 91-95.

Parthasarathy, S. & Rajendran, C. (1997a). A simulated annealing heuristic for scheduling to minimize mean weighted tardiness in a flowshop with sequence-dependent setup times of jobs-a case study. *Production Planning & Control* , 475-483.

Parthasarathy, S. & Rajendran, C. (1997b). An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *International journal of production economics* , 255-263.

Peterson, P. (2009). F2PY: a tool for connecting Fortran and Python programs. *International Journal of Computational Science and Engineering* , 296-305.

Rajendran, C. & Ziegler, H. (2003). Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. *European Journal of Operational Research* , 513-522.

Ruiz, R. & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research* , 1143-1159.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research* , 65-74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research* , 278-285.

	ssd10			ssd50			ssd100			ssd125		
	IG	IGS	TS	IG	IGS	TS	IG	IGS	TS	IG	IGS	TS
20 x 5	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	0,40	**0,02**	0,03	0,40	0,41	**0,00**
20 x 10	**0,00**	**0,00**	**0,00**	0,15	0,04	**0,00**	0,10	0,07	**0,00**	0,04	**0,00**	**0,00**
20 x 20	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
50 x 5	0,77	0,62	**0,60**	1,79	1,33	**0,85**	3,46	2,08	**1,68**	3,62	2,91	**1,86**
50 x 10	1,22	1,02	**0,64**	2,41	1,50	**0,96**	2,93	1,73	**1,20**	3,79	2,68	**1,91**
50 x 20	1,51	1,44	**1,17**	2,80	1,54	**1,18**	2,36	2,03	**1,12**	2,74	1,63	**0,91**
100 x 5	1,39	**1,07**	1,12	2,14	1,30	**1,15**	2,76	1,30	**0,69**	2,01	1,08	**0,60**
100 x 10	2,12	1,20	**1,14**	2,51	1,18	**0,91**	2,57	0,99	**0,96**	2,52	0,84	**0,68**
100 x 20	3,41	1,59	**1,28**	3,29	1,15	**0,91**	3,29	**1,19**	1,21	2,79	0,76	**0,30**
200 x 10	1,35	0,84	**0,59**	1,07	0,24	**0,05**	1,27	0,30	**0,20**	0,99	0,73	**0,72**
200 x 20	2,30	0,03	**-0,50**	1,61	-0,33	**-0,70**	1,60	-0,28	**-0,37**	1,17	-0,31	**-0,50**
500 x 20	0,66	-0,43	**-0,83**	0,95	-0,28	**-0,54**	0,63	0,76	**0,58**	**0,45**	1,59	1,38
average	1,23	0,62	**0,43**	1,56	0,64	**0,40**	1,78	0,85	**0,61**	1,71	1,03	**0,66**

Table 1: Relative percentage deviation errors obtained by Ruiz and Stützle (2008), by our implementation of their algorithm (IGS), and by tabu search (TS). Best results marked at bold.