



A local search method for permutation flow shop scheduling

WQ Huang and L Wang*

Huazhong University of Science and Technology, Wuhan, PR China

It is well known that a local search method, a widely used approach for solving the permutation flow shop scheduling problem, can easily be trapped at a local optimum. In this paper, we propose two escape-from-trap procedures to move away from local optima. Computational experiments carried out on a standard set of instances show that this heuristic algorithm generally outperforms an effective approximation algorithm.

Journal of the Operational Research Society (2006) **57**, 1248–1251. doi:10.1057/palgrave.jors.2602106

Published online 23 November 2005

Keywords: heuristics; local search; escape-from-trap; flow shop sequencing

Introduction

Flow shop scheduling, a well-known combinatorial optimization problem, has been proved to be NP-hard (Garey *et al*, 1976). The permutation flow shop scheduling problem (PFSP) studied in the present paper is scheduling n jobs on m machines (denoted as M_1, \dots, M_m) with the objective of minimizing the makespan, subject to the following four constraints: (i) each job is processed on all the machines according to the order of M_1, \dots, M_m , (ii) the processing time of each job on each machine is given, (iii) each machine can process at most one job at a time, and (iv) once a sequence $\sigma = (\sigma(1), \dots, \sigma(n))$ of all jobs is given by workers, each machine has to process all jobs according to sequence σ . In other words, the PFSP is to find a job permutation with minimum makespan. Given a sequence $\sigma = (\sigma(1), \dots, \sigma(n))$ of all jobs, let $p_{i,\sigma(j)}$ and $t_{i,\sigma(j)}$ denote the processing time of job $\sigma(j)$ on machine M_i and the start time of job $\sigma(j)$ on machine M_i , respectively, then the schedule can be generated by the following formulas:

$$t_{1,\sigma(1)} = 0$$

$$t_{1,\sigma(j)} = t_{1,\sigma(j-1)} + p_{1,\sigma(j-1)}, \quad 2 \leq j \leq n$$

$$t_{i,\sigma(1)} = t_{i-1,\sigma(1)} + p_{i-1,\sigma(1)}, \quad 2 \leq i \leq m$$

$$t_{i,\sigma(j)} = \max(t_{i-1,\sigma(j)} + p_{i-1,\sigma(j)}, t_{i,\sigma(j-1)} + p_{i,\sigma(j-1)}), \\ 2 \leq j \leq n, \quad 2 \leq i \leq m$$

Haouari and Ladhari (2003) presented the mathematical statement of the PFSP while Tseng *et al* (2004) discussed four integer programming models for the PFSP.

Moreover, Pan and Chen (2003) discussed re-entrant permutation flow-shop scheduling problem (RPFSP), which is an extension of the PFSP.

So far, many optimization algorithms and approximation algorithms have been proposed to solve PFSP. Optimization algorithms require too much computing time and cannot be accepted by workers.

On the other hand, approximation algorithms are widely used by workers, since they may give optimal or near optimal permutations quickly. These heuristic methods can be divided into two main classes: construction methods and improvement methods. The literature on construction methods, includes the heuristics proposed by Campbell *et al* (1970), Dannenbring (1977), Nawaz *et al* (1983) and Lai (1994); the method of Nawaz *et al* (NEH for short) is considered as the best one. Improvement methods such as simulated annealing algorithms proposed by Ogbu and Smith (1990) and Osman and Potts (1989) genetic algorithms proposed by Reeves (1995) and Chen *et al* (1995), and tabu search algorithms proposed by Nowicki and Smutnicki (1996), Widmer and Hertz (1989), Taillard (1989) and Grabowski and Pempera (2001) start from an initial permutation, which is usually generated by a construction method, and then iteratively generate a sequence of improved permutations. It is obvious that improvement methods generate significantly better solutions than construction ones.

In this paper, we present a new local search (LS) heuristic for the PFSP, which is based on a hybrid neighbourhood structure and two escape-from-trap procedures. Our method has been tested on 29 benchmark problem instances with up to 75 jobs and 20 machines. The experimental results of our method are compared with those of an effective hybrid heuristic (HH) (Wang and Zheng, 2003) that combined NEH heuristic with a genetic algorithm (GA) and simulated annealing (SA). For 10 problem instances named Rec13,

*Correspondence: L Wang, C5-211, Huazhong University of Science and Technology, Wuhan 430074, People's Republic of China.
E-mail: hustwanglei@sohu.com

Rec15, Rec17, Rec19, Rec21, Rec25, Rec27, Rec29, Rec37 and Rec41, our method generates better permutations than the HH. For three problem instances named Rec31, Rec33 and Rec39, our method generates worse permutations than the HH. For the rest of the problem instances, our method generates permutations whose makespans were equal to those of the permutations generated by the HH. So our method generally outperforms the HH.

The rest of the paper is organized as follows. In the next section, we state the neighbourhood structure, the escape-from-trap procedures, and the algorithm. Then we discuss computational results on a standard set of test problem instances. Finally, some concluding remarks are provided.

The strategy

The heuristic proposed in this paper is based on a LS procedure and two escape-from-trap procedures.

Neighbourhood structure

Given a permutation $\sigma = (\sigma(1), \dots, \sigma(n))$ of all jobs, the neighborhood of σ is defined as follows.

Insertion. Given a permutation σ , a neighbour σ' is generated by moving away any job from its position in permutation σ and inserting this job at any new position. Given a permutation $(J_4, J_2, J_5, J_1, J_3)$ of five jobs, each job can be inserted at four possible new positions. For example, job J_5 can be inserted before J_4 , or before J_2 , or after J_1 , or after J_3 so as to generate $(J_5, J_4, J_2, J_1, J_3)$, or $(J_4, J_5, J_2, J_1, J_3)$, or $(J_4, J_2, J_1, J_5, J_3)$, or $(J_4, J_2, J_1, J_3, J_5)$. Haouari *et al* (2003) discussed this kind of neighbourhood.

Block replacement. Given a permutation σ , a neighbour σ' is generated by replacing any k_1 successive jobs of permutation σ with any new sequence of these k_1 jobs, where k_1 is a given constant. Consider the following example. We suppose that $k_1 = 3$. Given a permutation $(J_4, J_2, J_5, J_1, J_3)$ of five jobs, a sequence (J_2, J_5, J_1) , which consists of three successive jobs of σ , can be replaced by (J_2, J_1, J_5) , or (J_5, J_2, J_1) , or (J_5, J_1, J_2) , or (J_1, J_2, J_5) , or (J_1, J_5, J_2) so as to generate $(J_4, J_2, J_1, J_5, J_3)$, or $(J_4, J_5, J_2, J_1, J_3)$, or $(J_4, J_5, J_1, J_2, J_3)$, or $(J_4, J_1, J_2, J_5, J_3)$, or $(J_4, J_1, J_5, J_2, J_3)$.

Escape-from-trap procedures

We suppose that $\sigma = (\sigma(1), \dots, \sigma(n))$ is a trap, that is, a local optimal permutation. Two escape-from-trap procedures are proposed in this paper.

Insertion. At first, job $\sigma(i)$, which is chosen uniformly at random from all jobs, is removed from its position in permutation σ . Then a new position of job $\sigma(i)$ is chosen uniformly at random from all positions. Finally, job $\sigma(i)$ is

inserted at this new position. This process is repeated c_1 times so as to move away from the local optimum, where c_1 is a given constant.

Block replacement. At first, a block b , which consists of k_2 successive jobs of permutation σ , is chosen uniformly at random from all blocks each of which consists of k_2 successive jobs of permutation σ , where k_2 is a given constant. Then a new block b' is chosen uniformly at random from all sequences of these k_2 successive jobs. Finally, block b is replaced with block b' so as to generate a new permutation of all jobs, that is, to move away from the local optimum.

In order to illustrate the escape-from-trap procedures, let us consider the following example. We suppose that $(J_5, J_1, J_2, J_4, J_3)$ is a local optimal permutation.

Insertion: We suppose that $c_1 = 2$. At first, we choose job J_1 from all jobs, remove J_1 from its position in permutation $(J_5, J_1, J_2, J_4, J_3)$, and insert J_1 before J_4 . Then we choose job J_3 from all jobs, remove J_3 from its position in permutation $(J_5, J_2, J_1, J_4, J_3)$, and insert J_3 before J_2 . So a new permutation $(J_5, J_3, J_2, J_1, J_3)$ is generated.

Block replacement. We suppose that $k_2 = 4$. We replace (J_1, J_2, J_4, J_3) with (J_4, J_3, J_1, J_2) . So a new permutation $(J_5, J_4, J_3, J_1, J_2)$ is generated.

Algorithm

Let T be the escape-from-trap counter, and T_m the maximum number of escape-from-trap allowed. Set $T = 0$, $T_m = 1000$, $k_1 = 4$, $k_2 = 6$, and $c_1 = 5$. We set k_1 to be a small number so as to limit the size of the neighbourhood. When a local optimal solution is found, the job sequence is softly perturbed. So k_2 and c_1 are also set to be small numbers.

- Step 1:* A permutation σ of all jobs is generated at random. Go to 2.
- Step 2:* Examine the permutations in the neighbourhood of permutation σ one by one. If a better permutation σ' than σ is found, then $\sigma = \sigma'$, go to 2. If all the permutations in the neighbourhood of permutation σ are no better than σ , then go to 3.
- Step 3:* Set $T = T + 1$. If $T = T_m$, then stop. If the makespan of permutation σ is equal to that of the best permutation (if the best permutation's makespan of the current problem instance is already presented in the literature), then stop. Move away from the local optimum as follows. A real number r is chosen uniformly at random from $[0, 1]$. If $r \geq 0.5$, use the first escape-from-trap procedure (insertion) to move away from the local optimum. And if $r < 0.5$, use the second escape-from-trap procedure (block replacement) to move away from the local optimum. Go to 2.

Computational results

This LS algorithm combined with the escape-from-trap procedures is programmed in the C language and run on a Pentium III 500 Mhz personal computer. The algorithm has been tested on 29 problem instances, which can be found in the OR-Library (<http://www.ms.ic.ac.uk/info.html>) and which are divided into the following two classes:

- (a) Eight instances (named Car1–Car8) provided by Carrier (1978).
- (b) Twenty-one instances (named Rec01–Rec41) provided by Reeves (1995).

The instances of set (a) are the easiest instances of these 29 instances. Rec37, Rec39 and Rec41 are the most difficult instances of these 29 instances. Generally speaking, the smaller an instance is, the easier it is.

The makespan value of the optimal permutation is shown in Tables 1 and 2, except for Rec37, Rec39 and Rec41. The optimal permutations of Rec37, Rec39 and Rec41 are still unknown, so the best known lower bound value of the makespan value of the optimal permutation is shown in Table 2.

Both our algorithm and another HH presented by Wang and Zheng (2003), which is run on a Pentium II 366 Mhz

Table 1 Results for the problem instances of class (a)

Problem	n, m	C*	LS				Hybrid heuristic			
			BRE	ARE	WRE	ART	BRE	ARE	WRE	ART
Car1	11, 5	7038	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.28
Car2	13, 4	7166	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.30
Car3	12, 5	7312	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.36
Car4	14, 4	8003	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.35
Car5	10, 6	7720	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.31
Car6	8, 9	8505	0.00	0.00	0.00	0.01	0.00	0.04	0.76	0.36
Car7	7, 7	6590	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.17
Car8	8, 8	8366	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.29
MBRE				0.00				0.00		

BRE, best relative error; ARE, average relative error; WRE, worst relative error; ART, average running time; MBRE, mean best relative error.

Table 2 Results for the problem instances of class (b)

Problem	n, m	C*	LS				Hybrid heuristic			
			BRE	ARE	WRE	ART	BRE	ARE	WRE	ART
Rec01	20, 5	1247	0.00	0.02	0.16	5.57	0.00	0.14	0.16	1.32
Rec03	20, 5	1109	0.00	0.00	0.00	2.57	0.00	0.09	0.18	1.33
Rec05	20, 5	1242	0.24	0.24	0.24	10.4	0.00	0.29	1.13	1.34
Rec07	20, 10	1566	0.00	0.00	0.00	1.86	0.00	0.69	1.15	4.98
Rec09	20, 10	1537	0.00	0.00	0.00	2.64	0.00	0.64	2.41	4.90
Rec11	20, 10	1431	0.00	0.00	0.00	0.77	0.00	1.10	2.59	4.96
Rec13	20, 15	1930	0.00	0.09	0.26	46.4	0.36	1.68	3.06	10.7
Rec15	20, 15	1950	0.00	0.44	0.77	35.3	0.56	1.12	2.00	10.7
Rec17	20, 15	1902	0.00	0.11	0.37	24.2	0.95	2.32	3.73	10.7
Rec19	30, 10	2093	0.38	0.63	0.86	82.8	0.62	1.32	2.25	15.0
Rec21	30, 10	2017	0.69	1.41	1.44	74.2	1.44	1.57	1.64	15.0
Rec23	30, 10	2011	0.40	0.54	0.80	81.0	0.40	0.87	1.69	15.1
Rec25	30, 15	2513	0.24	1.11	1.83	127	1.27	2.54	3.98	33.6
Rec27	30, 15	2373	0.59	0.94	1.18	137	1.10	1.83	4.00	33.8
Rec29	30, 15	2287	0.44	0.80	1.18	141	1.40	2.70	4.20	33.8
Rec31	50, 10	3045	1.54	1.91	1.97	292	0.43	1.34	2.50	64.9
Rec33	50, 10	3114	0.13	0.47	0.83	228	0.00	0.78	0.83	63.1
Rec35	50, 10	3277	0.00	0.00	0.00	6.40	0.00	0.00	0.00	64.8
Rec37	75, 20	4890	3.60	4.19	4.68	1700	3.75	4.90	6.18	860
Rec39	75, 20	5043	2.50	2.90	3.11	1680	2.20	2.79	4.48	877
Rec41	75, 20	4910	3.50	3.93	4.38	1910	3.64	4.92	5.91	873
MBRE			0.67					0.91		

personal computer, are executed 20 times independently and the statistical results and comparisons are summarized in Tables 1 and 2. For each run on each instance, the relative error RE (%) was calculated by following formula: $(C - C^*) / C^* \times 100\%$, where C denotes the makespan value of the permutation found by the algorithm, and C^* denotes makespan value of the optimal permutation or best known lower bound value of it.

Table 1 shows that both of these two algorithms are able to find the optimal permutations to all the instances of class (a). Whereas Table 2 shows that the LS algorithm generally provides better solutions than HH for the instances of class (b): the mean best relative error (MBRE) of LS is 0.67%, while that of HH is 0.91%.

Our algorithm is run on a Pentium III 500 Mhz personal computer while HH is run on a Pentium II 366 Mhz personal computer. The speed of the former computer is slower than two times of that of the latter one. For 11 instances (Car1–Car8, Rec07, Rec11, and Rec35), our algorithm consumes less time than HH if both algorithms are run on the same computer. For other instances, our algorithm consumes more time than HH if both algorithms are run on the same computer.

Our algorithm generates, generally, no worse solutions than HH because the local search procedure is combined with random perturbation in our algorithm.

Conclusion

In this paper, a hybrid local search algorithm for the PFSP is presented. The experiments carried out on benchmark problems demonstrate that this algorithm is effective.

The local search method is widely used for solving the scheduling problems, and the escape-from-trap procedures presented in this paper enhances its efficiency. Because this hybrid algorithm is a method of generality, it can also be used for solving other hard combinatorial optimization problems.

Acknowledgements—This research was supported by the National Natural Science Foundation of China under grant no. 10471051 and partially supported by the National Basic Research Program of China under grant no. 2004CB318000.

References

- Campbell HG, Dudek RA and Smith ML (1970). A heuristic algorithm for the n job m machine sequencing problem. *Mngt Sci* **16B**: 630–637.
- Carlier J (1978). Ordonnancements a contraintes disjonctives. *RAIRO—Opns Res* **12**: 333–351.
- Chen CL, Vempati VS and Aljaber N (1995). An application of genetic algorithms for flow shop problems. *Eur J Opl Res* **80**: 389–396.
- Dannenbring DG (1977). An evaluation of flowshop sequencing heuristics. *Mngt Sci* **23**: 1174–1182.
- Garey MR, Johnson DS and Sethi R (1976). The complexity of flow shop and job shop scheduling. *Math Opns Res* **1**: 117–129.

- Grabowski J and Pempera J (2001). New block properties for the permutation flowshop problem with application in tabu search. *J Opl Res Soc* **52**: 210–220.
- Haouari M and Ladhari T (2003). A branch-and-bound-based local search method for the flow shop problem. *J Opl Res Soc* **54**: 1076–1084.
- Lai TC (1994). A note on the heuristic flowshop scheduling. *Opns Res* **44**: 648–652.
- Nawaz M, Enscore EE and Ham I (1983). A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem. *Omega* **11**: 91–95.
- Nowicki E and Smutnicki C (1996). A fast tabu search algorithm for the permutation flow-shop problem. *Eur J Opl Res* **91**: 160–175.
- Ogbu FA and Smith DK (1990). Simulated annealing for the permutation flowshop problem. *Omega* **19**: 64–67.
- Osman IH and Potts CN (1989). Simulated annealing for permutation flow-shop scheduling. *Omega* **17**: 551–557.
- Pan JH and Chen JS (2003). Minimizing makespan in re-entrant permutation flow-shops. *J Opl Res Soc* **54**: 642–653.
- Reeves CR (1995). A genetic algorithm for flowshop sequencing. *Comput Opns Res* **22**: 5–13.
- Taillard E (1989). Some efficient heuristic methods for the flow shop sequencing problem. *Eur J Opl Res* **47**: 65–74.
- Tseng FT, Stafford EF and Gupta J (2004). An empirical analysis of integer programming formulations for the permutation flow shop. *OMEGA* **32**: 285–293.
- Wang L and Zheng DZ (2003). An effective hybrid heuristic for flow shop scheduling. *Int J Adv Manuf Tech* **21**: 38–44.
- Widmer M and Hertz A (1989). A new heuristic method for the flow shop sequencing problem. *Eur J Opl Res* **41**: 186–193.

Appendix

Notation

- n number of jobs
- m number of machines
- C^* makespan value of the optimal permutation, or best known lower bound value of it
- RE relative error to C^* , that is $(C - C^*) / C^* \times 100\%$, where C denotes the makespan value of the permutation found by the algorithm
- BRE best relative error, that is $(C_{\text{best}} - C^*) / C^* \times 100\%$, where C_{best} denotes the makespan value of the best permutation found by the algorithm
- ARE average relative error, that is $(C_{\text{average}} - C^*) / C^* \times 100\%$, where C_{average} denotes the average value of the makespans of the permutations found by the algorithm
- WRE worst relative error, that is $(C_{\text{worst}} - C^*) / C^* \times 100\%$, where C_{worst} denotes the makespan value of the worst permutation found by the algorithm
- MBRE mean best relative error
- ART average running time (s)

Performance measures evaluated over 20 runs for a given problem instance.

Received April 2004;
accepted June 2005 after two revisions