Int J Adv Manuf Technol (2008) 37:1200–1208
DOI 10.1007/s00170-007-1053-5

ORIGINAL ARTICLE

Performance evaluation of the scatter search method for permutation flowshop sequencing problems

M. Saravanan · A. Noorul Haq · A. R. Vivekraj · T. Prasad

Received: 7 August 2006 / Accepted: 12 April 2007 / Published online: 15 June 2007
© Springer-Verlag London Limited 2007

Abstract Many optimization problems from the industrial engineering world, in particular manufacturing systems, are very complex in nature and are quite hard to solve by conventional optimization techniques. There has been increasing interest to apply metaheuristic methods to solve such kinds of hard optimization problems. In this work, a novel metaheuristic approach called scatter search (SS) is applied for the $n/m/P/C_{\max}$ problem, an NP-hard sequencing problem, which is used to find a processing order of n different jobs to be processed on m machines in the same sequence with minimizing the makespan. SS contrasts with other evolutionary procedures by providing a wide exploration of the search space through intensification and diversification. In addition, it has a unifying principle for joining solutions and they exploit the adaptive memory principle to avoid generating or incorporating duplicate solutions at various stages of the problem. In this paper, various metaheuristic methods and best heuristics from the literature are used for solving the well-known benchmark problem set of Taillard (Eur J Oper Res 64:278–285, 1993). The results available for the various existing metaheuristic and heuristic methods are compared with the results obtained by the SS method. The proposed framework achieves better results for 4 of 12 benchmark problems and also achieves an average deviation of 1.003% from the benchmark problem set of Taillard (Eur J Oper Res 64:278–285, 1993). The computational results show that SS is a more effective metaheuristic for the $n/m/P/C_{\max}$ problem.

Keywords Permutation flowshop · Scheduling · Metaheuristic · Heuristic methods · Scatter search · Makespan

M. Saravanan (✉)
Department of Mechanical Engineering,
R.V.S. College of Engineering & Technology,
Dindigul 624005 Tamil Nadu, India
e-mail: sarandgl2k@yahoo.co.in

A. Noorul Haq · A. R. Vivekraj · T. Prasad
Department of Production Engineering,
National Institute of Technology,
Tiruchirappalli, Tamil Nadu, India

1 Introduction

A flowshop is characterized by the unidirectional flow of work with a variety of jobs being processed sequentially in a one-pass manner. A number of operations need to be done on every job in many manufacturing and assembly facilities. A "job" is, thus, a collection of operations to be performed on an item or unit with applicable technological constraints. This implies that all of the jobs have to follow the same route, even if the jobs are identical. The machines are assumed to be set up in a series and such a processing environment is referred to as a flowshop. In a scheduling problem using m machines, they must finish a total of n jobs and each job has exactly m operations, each of which must be proceeded in a different machine. Thus, each job has to pass through each machine in a particular order. The order of machines needed to complete a job is the same for all of the jobs. The objective of the scheduling problem now is to determine a sequence on each machine that satisfies the above constraints and minimizes the objective function.

In this paper, the permutation flowshop problem has been attempted with a novel evolutionary technique called scatter search (SS). This algorithm incorporates procedures based on different strategies, such as diversification, local search, Tabu search, or path relinking. In common with

other evolutionary methods, SS operates with a population of solutions, rather than with a single solution at a time, and employs procedures for combining these solutions to create new ones. SS, in contrast with other evolutionary procedures such as genetic algorithms, provides an unifying principle for joining solutions based on generalized path constructions and by utilizing strategic designs, whereas other approaches resort to randomization. Additional advantages are provided by intensification and diversification mechanisms that exploit adaptive memory, together with processes to avoid generating or incorporating duplicate solutions at various stages and drawing on foundations that link SS and path relinking to Tabu search [1].

2 Literature survey

During the last 40 years, the $n/m/P/C_{\max}$ problem has held the attention of many researchers [2]. Although optimal solutions of $n/m/P/C_{\max}$ problems can be obtained via enumeration techniques such as exhaustive enumeration and the branch and bound method [3], these methods may take a prohibitive amount of computation time, even for moderate size problems. Sequencing methods in the literature can be broadly categorized into two types of approaches, namely, optimization and heuristic. Optimization approaches guarantee to obtain the optimum sequence, whereas heuristic approaches mostly obtain near-optimal sequences. Among the optimization approaches, the algorithm developed by Johnson [4] is the widely cited research dealing with sequencing n jobs on two machines. Lomnicki [5] proposed a branch and bound technique to find the optimum permutation of jobs. Since the flowshop scheduling problem has been recognized to be NP-hard, the branch and bound method cannot be applied for large size problems. This limitation has encouraged researchers to develop efficient heuristics. For practical purposes, it is often more appropriate to look for a heuristic method that generates a near-optimal solution at relatively minor computational expense. This leads to the development of many heuristic procedures.

The currently available heuristics for solving this problem in the literature can be classified into two categories: constructive heuristics and improvement heuristics [6]. In the constructive heuristic, once a job sequence is determined, it is fixed and cannot be reversed. In the constructive category, methods developed by Palmer [7], Campbell et al. [8], Gupta [9], Dannenbring [10], Rock and Schmidt [11], and Nawaz et al. [12] can be listed. Mostly, these methods are developed on the basis of Johnson's algorithm. Turner and Booth [13] and Taillard [14] have verified that the method proposed by Nawaz et al. [12], namely NEH, performs well among the constructive methods tested. On the other hand, Osman and Potts [6], Widmer and Hertz [2], Ho and Chang [15], Ogbu and Smith [16], Taillard [14], Nowicki and Smutnicki [17], and Ben-Daya and Al-Fawzan [18] have developed improvement heuristics for the same problem.

The improvement heuristics start with an initial solution and then provide a scheme for iteratively obtaining an improved solution. In recent years, studies with metaheuristics have been extensively carried out on this argument. The metaheuristic is a rather general algorithmic framework that can be applied to different optimization problems with minor modifications. Essentially, it is a type of randomized improvement heuristic [6]. Methods of this type include genetic algorithms [19, 20], simulated annealing [21, 22], and the Tabu search [23]. The literature shows that these methods can obtain very good results for NP-hard combinatorial optimization problems. Nowicki and Smutnicki [24] have developed a new algorithm called the modified scatter search algorithm (MSSA). MSSA produced 20 new, better upper bound solutions among 30 very hard, as yet unsolved instances from common benchmark sets. Another metaheuristic is given by Stutzle [25], called the iterated local search (ILS). According to the tests conducted by Stutzle, the ILS algorithm is much better than the Tabu search of Taillard [14] and is also better than the Tabu search of Nowicki and Smutnicki (TSAB) [17]. Ruiz and Maroto [26] compared 25 methods, ranging from the classical Johnson's algorithm or dispatching rules to the most recent metaheuristics, including Tabu search, simulated annealing, genetic algorithms, ILS, and hybrid techniques for the benchmark problems [27]. In their paper [26], all of the algorithms and methods are coded in Delphi 6.0 and run on an AthlonXP 1600+ computer with 512 MB of main memory. The methods used for comparison are well known metaheuristics, such as Osman and Potts' [6] SA algorithm (SAOP), Widmer and Hertz's [2] SPIRIT, Chen et al.'s [28] GA algorithm (GAChen), Reeves and Yamada's [29] GA algorithm (GAReev), hybrid GA+local search by Murata et al. [30] (GAMIT), Stutzle's ILS [25], and GA by Ponnambalam et al. [31] (GAPAC). Reza Hejazi and Saghafian [32] have described a complete survey of flowshop scheduling problems up to 2004. The proposed SS method outperforms other methods for the flowshop scheduling problems of Noorul Haq et. al. [33].

This paper is organized as follows. Section 3 describes the formulation of the sequencing problem. In Sect. 4, the elements of the SS method based on the sequencing problems are discussed. A simple problem is solved by the proposed SS method in Sect. 5. The computational results obtained by the application of this method to the problems selected from benchmark problems [27] are discussed in Sect. 6. Section 7 includes our discussions and conclusions.

3 Problem formulation

The permutation flowshop scheduling problem consists of scheduling n jobs with given processing times on m machines, where the sequence of processing a job on all machines is identical and unidirectional for each job. In studying flowshop scheduling problems, it is a common assumption that the sequence in which each machine processes all jobs is identical on all machines (permutation flowshop). A schedule of this type is called a permutation schedule and is defined by a complete sequence of all jobs. This paper addresses the same problem representation used in the literature [34].

3.1 Assumptions in permutation flowshop scheduling

The operating sequences of the jobs are the same on every machine and the common sequence has to be determined. The following assumptions are made for this work:

1. n jobs, $j=\{i/i=1, 2,..., n\}$, are available at time 0.
2. There are m machines $m_1, m_2,..., m_m$ to process n jobs, each having a sufficient capacity of buffer for work-in-process.
3. Each job can be processed by at most one machine at a time. Each machine can process at most one job at a time, and is never interrupted during processing.
4. The processing sequence of n jobs on each machine is the same, i.e., an optimal permutation schedule is sought.
5. The setup times of the operations are included in the processing time and do not depend on the sequence.

3.2 Permutation flowshop problem representation

The permutation flowshop represents a particular case of the flowshop scheduling problem, having as the goal the deployment of an optimal schedule for n jobs on m machines. Solving the flowshop problem consists of scheduling n jobs (i=1,..., n) on m machines (j=1,..., m). A job consists of m operations and the jth operation of each job must be processed on machine j. So, one job can start on machine j if it is completed on machine $j-1$ and if machine j is free. Each operation has a known processing time $p_{i,j}$. For the permutation flowshop, the operating sequences of the jobs are the same on every machine. If one job is at the ith position on machine 1, then this job will be at the ith position on all of the machines.

As a consequence, for the permutation flowshop problem, considering the makespan as the objective function to be minimized, solving the problem means determining the permutation which gives the smallest makespan value. In the above specified context, a job j_i can be seen as a set of operations, having one operation for each of the m machines:

- $j_i=\{O_{i1}, O_{i2}, O_{i3},..., O_{iM}\}$, where O_{ij} represents the jth operation of j_i
- Operation O_{ij} must be processed on machine m_j
- For each operation O_{ij} , there is an associated processing time p_{ij}

Notationally, $F/P/C_{max}$, considering the makespan as the objective function to minimize the overall processing time, refers the problem.

Let $\prod_1, \prod_2, \prod_3,..., \prod_N$ be a permutation. Computing the completion time $C(\prod_i, j)$ for the ith job of the given permutation $\prod$ and machine j can be done as follows:

$$
\begin{aligned}
&C(\Pi_1,\ 1) = p_{\Pi 1},\ 1 \\
&C(\Pi_i,\ 1) = C(\Pi_{i-1},\ 1) + p_{\Pi i},\ 1 && i = 2, \ldots,\ N \\
&C(\Pi_1,\ j) = C(\Pi_i,\ j-1) + p_{\Pi 1},\ j && i = 2, \ldots,\ M \\
&C(\Pi_i,\ j) = \max\{C(\Pi_{i-1},\ 1), C(\Pi_1,\ j-1)\} && i = 2, \ldots,\ N; \\
&\qquad + p_{\Pi i},\ j && j = 2, \ldots,\ M
\end{aligned}
$$

Under these specifications, the value of the objective function, the makespan, C_{max}, is given as $C(\prod_N, M)$ −completion time for the last operation on the last machine.

4 Elements of scatter search

The solution approach that is developed for this permutation problem consists of an adaptation of the scatter search (SS) method. SS is an instance of the so-called evolutionary method, which is not based solely on randomization as the main mechanism for searching. It constructs solutions by combining others by means of strategic designs that exploit the knowledge on the problem at hand. The goal of these procedures is to enable a solution procedure based on the combined elements to yield better solutions than one based on the original elements.

Compared to other evolutionary methods, SS operates with a population of solutions, rather than with a single solution at a time, and employs procedures for combining these solutions to create new ones. The meaning of "combining" and the motivation for carrying it out has a rather special origin and character in the SS setting. One of the distinguishing features of this approach is its intimate association with the Tabu search (TS) metaheuristic, and, hence, its adoption of the principle that searching can benefit by incorporating special forms of adaptive memory, along with procedures particularly designed for exploiting that memory. More about the origin and multiple applica-

tions of SS can be found in Glover et al. [1]. The basic steps involved in the static SS are:

Step 1 Use the diversification generator to generate diverse trial solutions from the seed solution(s)
Step 2 Use the improvement method to create one or more enhanced trial solutions
Step 3 With these initial solutions, update the reference set (RefSet)
Step 4 Repeat

4.1 Generate subsets of the RefSet
4.2 Combine these subsets and obtain new solutions
4.3 Use the improvement method to create a more enhanced trial solution
4.4 While continuing to maintain and update the RefSet

Until Refset is stable (no new solutions are included)
Step 5 If iterations (steps 1–4) elapse without improvement, stop. Else, return to step 1.

4.1 Diversification generation method

The diversification generation method is used to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input. This element of the SS approach is particularly important, given the goal of developing a method that balances diversification and intensification in the search. This method was suggested by Glover [35], which generates diversified permutations in a systematic way without reference to the objective function. Assume that a given trial permutation (P) used as a seed is represented by indexing its elements, so that they appear in consecutive order, to yield P=(1, 2,..., n). Define the subsequence $P(h: s)$, where s is a positive integer between 1 and h, to be given by $P(h: s)$=(s, s+h, s+2h,..., s+rh), where r is the largest nonnegative integer such that $s+rh \leq n$. Then, define the permutation $P(h)$, for $h \leq n$ to be:

$$P(h) = (P(h:\ h),\ P(h:\ h-1), \ldots,\ P(h:\ 1))$$

To illustrate:

– Suppose P is given by P=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12). If h=4 is chosen, then:

$$P(4:\ 4) = (4,\ 8,\ 12)$$
$$P(4:\ 3) = (3,\ 7,\ 11)$$
$$P(4:\ 2) = (2,\ 6,\ 10)$$
$$P(4:\ 1) = (1,\ 5,\ 9)$$

to give:

$$P(4) = (4,\ 8,\ 12,\ 3,\ 7,\ 11,\ 2,\ 6,\ 10,\ 1,\ 5,\ 9)$$

– Similarly, if h=3 is chosen, then:

$$P(3:\ 3) = (3,\ 6,\ 9,\ 12)$$
$$P(3:\ 2) = (2,\ 5,\ 8,\ 11)$$
$$P(3:\ 1) = (1,\ 4,\ 7,\ 10)$$

to give:

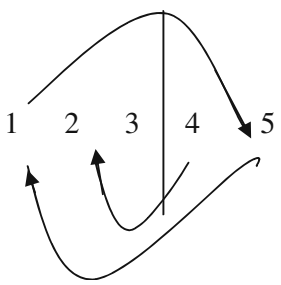
$$P(3) = (3,\ 6,\ 9,\ 12,\ 2,\ 5,\ 8,\ 11,\ 1,\ 4,\ 7,\ 10)$$

In this illustration, h is allowed to take the two values closest to the square root of n. These values are interesting based on the fact that, when h equals the square root of n, the minimum relative separation of each element from every other element in the new permutation is maximum, compared to the relative separation of exactly 1 in the permutation P. In general, for the goal of generating a diverse set of permutations, preferable values for h range from 1 to n/2.

4.2 Improvement method

The improvement method is used to transform a trial solution into one or more enhanced trial solutions. Neither the input nor the output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If there is no improvement in the input trial solution results, the "enhanced" solution is considered to be the same as the input solution. For each trial solution produced in diversification, we use the improvement method to create one more enhanced trial solution.

For example:



The sequence is divided into two by taking half the number of jobs on both sides. If the number of jobs is not an even number, more than half the number of jobs is taken on the left side (refer to the above example). The jobs on the right side of the sequence have to be inserted on the left

side. Similarly, the jobs on the left side of the sequence have to be inserted on the right side. The improved trial solution will be obtained by the improvement method.

4.3 Reference set update method

The reference set (RefSet) update method accompanies each application of the improvement method, and is generally examined straight after the improvement method, because of its linking role with the subset generation method. The update operation consists of maintaining a record of the b best solutions found, where the value of b is treated as a constant search parameter. The issues associated with this updating function are conceptually straightforward; to build and maintain a reference set consisting of the b best solutions is found (where the value of b is typically small, e.g., not more than 20), organized to provide efficient accessing by other parts of the method. Solutions gain membership to the reference set according to their quality or their diversity.

4.4 Subset generation method

This procedure consists of creating different subsets X of RefSet, as a basis for implementing the subsequent combination method. The SS methodology prescribes that the set of combined solutions (i.e., the set of all combined solutions that the implementation intends to generate) is produced in its entirety at the point where X is created.

Therefore, once a given subset X is created, there is no merit in creating it again. The procedure seeks to generate subsets X of RefSet that have useful properties, while avoiding the duplication of subsets previously generated. The approach for doing this is organized to generate four different collections of subsets of RefSet, subset type 1, 2, 3, and 4, with the following characteristics:

- Subset type=1: all two-element subsets
- Subset type=2: three-element subsets derived from the two-element subsets by augmenting each two-element subset to include the best solution not in this subset
- Subset type=3: four-element subsets derived from the three-element subsets by augmenting each three-element subset to include the best solutions not in this subset
- Subset type=4: the subsets consisting of the best i elements, for i=5 to b

A central consideration of this design is that RefSet itself might not be static, because it might be changing as new solutions are added to replace old ones (when these new solutions qualify to be among the current b best solutions found). In the implementation, however, a static updating of RefSet is maintained, but a broad definition of "best" for the membership in this set is used. In other words, RefSet is not allowed to dynamically change its size, but two criteria are used to allow solutions initially become members of this set. One criterion is the quality of the solution (as given by the objective function value) and the other is the diversity of the solution (as given by the dissimilarity measure). In this sense, the definition of "best" to construct the first RefSet is broader than one that considers only the value of the objective function. After the first RefSet has been created, subsequent membership in the set can only be obtained by means of solution quality. That is, RefSet changes when the combination method generates solutions of higher quality and the process stops when RefSet converges.

4.5 Solution combination method

This method is used to transform a given subset of solutions produced by the subset generation method into one or more combined solution vectors. Specific processes for carrying out these steps are described by Martí et al. [36]. Here, the solution combination method, which is applied to each subset generated in the previous step, uses a min–max construction based on votes. The method scans (from left to right) each reference permutation in the subset, and uses the rule that each reference permutation votes for its first element that is still not included in the combined permutation (referred to as the "incipient element"). The voting determines the next element to enter the first still unassigned position of the combined permutation. This is a min–max rule in the sense that, if any element of the reference permutation were chosen other than the incipient element, then it would increase the deviation between the reference and the combined permutations. Similarly, if the incipient elements were placed later in the combined permutation than its next available position, this deviation would also increase. So the rule attempts to minimize the maximum deviation of the combined solution from the reference solution, subject to the fact that other reference solutions in the subset are also competing to contribute.

5 Numerical illustration

In this illustration, the processing times for a four-job and three-machine problem is given below:

Job	Machine			
	*	M1	M2	M3
	J1	1	8	4
	J2	2	4	5
	J3	6	2	8
	J4	3	9	2

The detailed steps involved in the proposed SS method are given as follows:

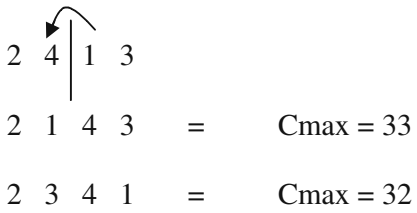
Step 1 Generate trial schedules by using the NEH [12] method as the seed sequence, say (1, 2, 3, 4)

Step 2 Use the diversification generation method to generate diverse trial solutions from the seed solution.

Select the value of h for the seed sequence. The preferable value is 1 to $n/2$, since n=4, h=2:

$$P(2:2) = (2, 4)$$
$$P(2:1) = (1, 3)$$
$$P(2) = (2, 4, 1, 3) = C_{\max} \text{ value is } 35$$
$$\text{if } h = 1,\ P(1) = (1, 2, 3, 4) = C_{\max} = 28$$

Step 3 For each trial, with trial solutions produced in step 2, use the improvement method to create one or more enhanced trial solutions. During successive applications of the step, maintain and update a reference set containing the b best solutions. In the improvement method, the insertion of each element on both sides is carried out, as explained in Sect. 4.2. Example:



Likewise, left to right insertions should be carried out. Finally, the best sequence is selected and, again, the same procedure is repeated until there is no improvement observed.

Step 4 From the improvement results, rank the sequence in order of increasing makespan and form the RefSet consists of the best and diverse solutions (refer to Sect. 4.3):

$$\left.\begin{array}{l}\text{RefSet}(1) = 2\,1\,4\,3\\ \text{RefSet}(2) = 2\,4\,3\,1\end{array}\right\}\text{best solutions}$$

$$\left.\begin{array}{l}\text{RefSet}(3) = 1\,4\,2\,3\\ \text{RefSet}(4) = 3\,1\,4\,2\end{array}\right\}\text{diverse solutions}$$

Step 5 The subsets are formed as:

$$\begin{aligned}\text{Subset } 1 &= (1, 2)(1, 3)(1, 4)\\ &\quad (2, 3)(2, 4)(3, 4)\\ \text{Subset } 2 &= (1, 2, 3)(1, 3, 4)\\ &\quad (2, 3, 4)\\ \text{Subset } 3 &= (1, 2, 3, 4)\end{aligned}$$

(Similarly, all of the four types of subsets are formed for big problems. Refer to Sect. 4.4.)

6. For each subset produced in step 5, the solution combination method (refer to Sect. 4.5) is carried out as follows, e.g.:

$$\text{Subset } 1 = (1, 2)$$
$$\text{RefSet}(1) = 2143$$
$$\text{RefSet}(2) = 2431$$
$$\text{Combined sequence} = 2143 = C_{\max} = 33$$

Similarly, all of the sets are combined.

The best solution from the combination set is added to the RefSet. The same procedure is repeated for the new updated RefSet until the best possible objective value is obtained.

6 Computational results

One difficulty faced by researchers in scheduling is to compare their developed heuristics with those of other researchers. If the standard set of test problems is accessible, different algorithms' performances can be compared on exactly the same set of test problems. For this reason, 120 benchmark problems are chosen from Taillard [27] as the test problems for this study. Taillard has produced a set of $n/m/P/C_{\max}$ problems with 5, 10, and 20 machines and from 20 to 500 jobs .The instances were randomly generated as follows for each job i (i=1, 2,..., n) on each machine j (j=1, 2,..., m), and an integer processing time p_{ij} was generated from the uniform distribution [1, 99]. In order to propose problems that are as difficult as possible, Taillard [27] generated many instances of problems, then, for each size of problems, chose the ten instances that seemed to be the hardest ones to form a basic problems set. Thus, there were ten instances for each problem size and 120 problem instances in all. Subsequently, Taillard [27] gave each of these instances with the following information: initial value of the random generator's seed, a lower bound, and an upper bound of the optimal makespan. The test problem files are available via Taillard's website at http://ina2.eivd.ch/collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html.

The algorithm was coded using C++ and run on an Intel Pentium IV, 3-GHz PC. To evaluate the algorithm, each of the problem instances was tested for ten trials. The best trial was chosen and the ten instances for the same problem size were averaged. The final results are shown in Table 1, which gives a comparison with other metaheuristics. Ant colony systems (ACS) has been discussed by Ying and Yiao [37], genetic algorithms (GA), simulated annealing (SA), and neighborhood search (NS) methods were discussed by Colin [34], MSSA was discussed by Nowicki and Smutnicki [24], various metaheuristics were discussed by Ruiz and Maroto [26], and the best heuristics were

Table 1 Comparison of the scatter search (SS) method with various metaheuristic and heuristic algorithms for permutation flowshop problems adopted from Taillard [27]

Problem size	Scatter search		Metaheuristic methods											Heuristic methods			
(Jobs/ machines)	PRD	CPU time (s)	ACS Quality	GA	SA	NS	MSSA	SAOP	Spirit	GA-Chen	GA-Reev	GA-MIT	ILS	GAPAC	Palmer	CDS	NEH	Ho-Cha
20/5	0.33	3	1.19	1.61	1.27	1.46	–	1.39	5.22	3.82	0.70	4.21	**0.24**	8.98	10.58	9.54	3.35	6.94
20/10	0.88	4	1.70	2.29	1.71	2.02	–	2.66	5.86	4.89	1.92	5.40	**0.77**	13.61	15.28	12.13	5.02	10.51
20/20	**0.497**	3	1.60	1.95	0.86	1.10	–	2.31	4.58	4.17	1.53	4.53	0.85	11.03	16.34	9.64	3.73	8.30
50/5	0.166	6	0.43	0.45	0.78	0.79	–	0.69	2.03	2.09	0.26	3.11	**0.12**	6.50	5.34	6.10	0.84	3.33
50/10	**1.561**	8	1.89	2.28	1.98	3.21	–	4.25	5.88	6.60	2.58	8.38	2.01	16.41	14.01	12.98	5.12	11.29
50/20	3.117	11	2.71	3.44	2.86	3.90	**−0.82**	5.13	7.21	8.03	3.76	10.65	3.29	18.56	15.99	13.85	6.20	12.40
100/5	**0.09**	44	0.22	0.23	0.56	0.76	–	0.40	1.06	1.32	0.18	5.41	0.11	5.32	2.38	5.01	0.46	2.70
100/10	0.73	52	1.22	1.25	1.33	2.69	–	1.88	5.07	3.75	1.08	12.05	**0.66**	12.34	9.20	9.15	2.13	7.96
100/20	1.477	53	2.22	2.91	2.32	3.98	**−1.15**	5.21	10.15	7.94	3.94	18.24	3.17	18.25	14.41	13.12	5.11	11.10
200/10	**0.44**	96	0.64	0.50	0.83	3.81	–	1.56	9.03	2.70	0.82	7.52	0.49	9.75	5.13	7.38	1.43	5.11
200/20	1.26	94	1.30	1.35	1.74	6.07	**− 1.398**	4.83	16.17	7.07	3.33	15.35	2.74	17.06	13.17	12.08	4.37	9.99
500/20	1.49	129	1.68	**−0.22**	0.85	9.07	–	3.40	13.57	4.61	1.83	2.17	1.29	2.61	7.09	8.55	2.24	7.14
Average	**1.003**	60.36	1.40	1.50	1.42	3.24	– 1.12	2.81	7.15	4.75	1.83	8.92	1.31	12.53	10.74	9.96	3.33	8.06

discussed in the same paper [26]. The solution quality is measured by the mean percentage difference from Taillard's upper bound. The SS algorithm needs an initial permutation, which can be found by any method. In the tests, which are conducted in a similar fashion as they are carried out in previous papers [17, 37], algorithm NEH [12], which is considered to be the best among simple constructive heuristics for flowshop scheduling, is used. For each test instance, the makespan (C_s) using the SS algorithm is found. The percentage relative deviation (PRD) is used to measure the algorithm quality:

$$PRD = 100\frac{(C_s - C_T)}{C_T}$$

The PRD is the percentage relative difference between makespan C_s and the reference makespan C_T given by Taillard [27]. Since the CPU times vary according to the hardware, software, and coding, the computational efficiency cannot be compared directly in this paper. When applying heuristics, the properties of reliability and consistency are both important. As demonstrated, SS is very consistent in the solution quality, with respect to the chance of obtaining a bad solution, and the computational time. It is clear that SS is promising for the $n/m/P/C_{\max}$ problem. On the whole, SS outperforms the various other metaheuristics and heuristic methods except for MSSA. The PRD is found to be 1.003%, with a maximum of 3.117%.

7 Conclusion

The main aim of this research is to explore the potential of the scatter search (SS) for scheduling problems of a permutation flowshop. The inherent weakness of many search procedures is that they often get trapped in a region around some local minima. Their ability to break out of such entrapments and achieve better, ideally global minima, is based on their capacity to provide a suitable mixture of intensification and diversification. SS also provides unifying principles for joining solutions based on generalized path constructions and by utilizing strategic designs where other approaches resort to randomization. The SS algorithm is tested on problems adopted from Taillard [27] and compared with the other metaheuristic methods used and compared by Ying and Liao [37] and Ruiz and Maroto [26]. In fact, the SS metaheuristic can achieve an average deviation of 1.003% from the upper bound solution of the benchmark problems of Taillard [27]. The experimental observations verified the effectiveness and efficiency of the SS algorithm over the other metaheuristics. From the results given in Table 1, it is concluded that the computational time required to obtain the optimal solution is comparatively low.

References

1. Glover F, Laguna M, Marti R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29(3):653–684
2. Widmer M, Hertz A (1989) A new heuristic method for the flow shop sequencing problem. Eur J Oper Res 41(2):186–193
3. Pinedo M (1995) Scheduling: theory, algorithm, and systems. Prentice-Hall, Englewood Cliffs, New Jersey
4. Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included. Nav Res Logist Q 1(1):61–68
5. Lomnicki ZA (1965) A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. Oper Res Q 16(1):89–100
6. Osman IH, Potts CN (1989) Simulated annealing for permutation flow-shop scheduling. OMEGA Int J Manag Sci 17(6):551–557
7. Palmer DS (1965) Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. Oper Res Q 16(1):101–107
8. Campbell HG, Dudek RA, Smith ML (1970) A heuristic algorithm for the *n*-job, *m*-machine sequencing problem. Manage Sci 16(10):630–637
9. Gupta JND (1971) A functional heuristic algorithm for the flowshop scheduling problem. Oper Res Q 22(1):39–47
10. Dannenbring DG (1977) An evaluation of flow shop sequencing heuristics. Manage Sci 23(11):1174–1182
11. Rock H, Schmidt G (1982) Machine aggregation heuristics in shop scheduling. Methods Oper Res 45:303–314
12. Nawaz M, Enscore EE Jr, Ham I (1983) A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. OMEGA Int J Manag Sci 11(1):91–95
13. Turner S, Booth D (1987) Comparison of heuristics for flow shop sequencing. OMEGA Int J Manag Sci 15(1):75–78
14. Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47(1):65–74
15. Ho JC, Chang Y-L (1990) A new heuristic for the *n*-job, *m*-machine flow shop problem. Eur J Oper Res 52(2):194–206
16. Ogbu FA, Smith DK (1990) The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flow shop problem. Comput Oper Res 17(3):243–253
17. Nowicki E, Smutnicki C (1996) A fast tabu search algorithm for the permutation flow-shop problem. Eur J Oper Res 91(1):160–175
18. Ben-Daya M, Al-Fawzan M (1998) A tabu search approach for the flow shop scheduling problem. Eur J Oper Res 109(1):88–95
19. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, Massachusetts
20. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, Michigan
21. van Laarhoven PJM, Aarts EHL (1987) Simulated annealing: theory and applications. D. Reidel, Dordrecht, The Netherlands
22. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculation by fast computing machine. J Chem Phys 21(6):1087–1092
23. Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Boston, Massachusetts
24. Nowicki E, Smutnicki C (2006) Some aspects of scatter search in the flow-shop problem. Eur J Oper Res 169(2):654–666
25. Stutzle T (1998) Applying iterated local search to the permutation flow shop problem. Technical report, AIDA-98-04, FG Intellektik, TU Darmstadt, Germany
26. Ruiz R, Maroto C (2005) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165(2):479–494
27. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
28. Chen C-L, Vempati VS, Aljaber N (1995) An application of genetic algorithms for flow shop problems. Eur J Oper Res 80(2):389–396

29. Reeves C, Yamada T (1998) Genetic algorithms, path relinking, and the flowshop sequencing problem. Evol Comput 6(1):230–234
30. Murata T, Ishibuchi H, Tanaka H (1996) Genetic algorithms for flowshop scheduling problems. Comput Ind Eng 30(4):1061–1071
31. Ponnambalam SG, Aravindan P, Chandrasekaran S (2001) Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. Prod Plan Control 12(4):335–344
32. Reza Hejazi S, Saghafian S (2005) Flowshop-scheduling problems with makespan criterion: a review. Int J Prod Res 43(14): 2895–2929
33. Noorul Haq A, Saravanan M, Vivekraj AR, Prasad T (2007) A scatter search approach for general flowshop scheduling problem. Int J Adv Manuf Technol 31(7–8):731–736
34. Colin RR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22(1):5–13
35. Glover F (1998) A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) Lecture notes in computer science, vol 1363, pp 13–54 (expanded version available on request)
36. Martí R, Laguna M, Campos V (2005) Scatter search vs. genetic algorithms: an experimental evaluation with permutation problems. In: Rego C, Alidaee B (eds) Metaheuristic optimization via adaptive memory and evolution: tabu search and scatter search. Kluwer Academic Publishers, Norwell, Massachusetts, pp 263–282
37. Ying K-C, Liao C-J (2004) An ant colony system for permutation flow-shop sequencing. Comput Oper Res 31(5):791–801