An Ant-Colony Algorithm to Transform Jobshops into Flowshops: A Case of Shortest-Common-Supersequence Stringology Problem

Suchithra Rajendran[1], Chandrasekharan Rajendran[2], and Hans Ziegler[3]

[1] Department of Industrial Engineering, College of Engineering, Guindy, Anna University, Chennai 600025, India
[2] Department of Management Studies, Indian Institute of Technology Madras, Chennai 600036, India
[3] Department of Production and Logistics, University of Passau, 94032 Passau, Germany
snowy_410@yahoo.co.in, craj@iitm.ac.in, ziegler@uni-passau.de

Abstract. In this work we address the problem of transforming a jobshop layout into a flowshop layout with the objective of minimizing the length of the resulting flowline. This problem is a special case of the well-known classical Shortest Common Supersequence (SCS) stringology problem. In view of the problem being NP-hard, an ant-colony algorithm, called PACO-SFR, is proposed. A new scheme of forming an initial supersequence of machines (i.e., flowline) is derived from a permutation of jobs, followed by the reduction in the length of the flowline by using a concatenation of forward reduction and inverse reduction techniques, machine elimination technique and finally an adjacent pair-wise interchange of machines in the flowline. The proposed ant-colony algorithm's performance is relatively evaluated against the best known results from the existing methods by considering many benchmark jobshop scheduling problem instances.

Keywords: Jobshop, Flowshop, Shortest Common Supersequence, Ant-colony algorithm.

1 Introduction

A jobshop is a manufacturing system that has a process layout with machines capable of performing similar operations located together, while a flowshop is a manufacturing system that has a flowline-based (i.e., product-based) layout such that products or jobs move in the shop with a uni-directional flow in the order of their processes. In other words, the flow of all jobs through the shop for processing on machines is in the same forward direction with no back-tracking, but it is possible that a job may skip some machines for its processing in the flowline. According to Knolmayer et al. [1], a jobshop layout is a common configuration in many manufacturing systems, and the transformation of jobshops into flowshops is vital and relevant in the context of an efficient supply chain management. Kimms [2] observed that while transforming a jobshop layout into a flowshop, the key objective is to minimize the length of the

J. Suzuki and T. Nakano (Eds.): BIONETICS 2010, LNICST 87, pp. 413–424, 2012.

resultant flowshop because the minimization of the length of the flowshop serves to minimize the lead times of production of jobs in the resultant flowshop, thereby leading to reduced inventory levels.

The problem of minimizing the length of the resultant flowline is a special case of the well-known classical Shortest Common Supersequence (SCS) stringology problem [2-4]. Framinan [4] observed that even though the SCS problem and the problem of transforming a jobshop into a flowshop appear identical in terms of complexity and problem statement, they are not, in general, equivalent with respect to some of the assumptions in jobshop problems (e.g. in a jobshop it is assumed that the consecutive operations of a job are not performed on the same machine; mostly it is assumed that each job is processed on each machine only once and a job is processed on all machines, and the sequence of processing a job is independent of the sequence of processing another job). The general SCS problem is known to be NP-hard [5-6] and so is the problem under consideration [2]. Some attempts towards transforming a jobshop into a resultant flowshop are due to Kimms [2], Framinan and Ruiz-Usano [3] and Framinan [4, 7]. Framinan [4] made a thorough analysis of the existing algorithms for the SCS problem (e.g. genetic algorithms by Branke et al. [8]), ant colony algorithm by Michel and Middendorf [9] and beam search by Framinan and Ruiz-Usano [3] adapted to the jobshop-transformation problem under consideration and the proposed tabu search by considering seventy well-known jobshop problem instances, and reported the findings (see Table 6 of that paper) in order to be used as a benchmark for future researchers. To our knowledge, the work by Framinan [4] is the most exhaustive study till date.

In the present work we propose an ant-colony algorithm with some new features for transforming a jobshop into a flowshop. First we construct an initial supersequence of machines on the basis of a sequence of jobs and the associated machine ordering, followed by two concatenations of forward and inverse reduction procedures, machine elimination technique and finally a local search scheme involving an adjacent pair-wise interchange of machines in the flowline, applied twice. Our work employs an approach of obtaining the initial supersequence of machines that is different from the previous attempts (e.g. Framinan and Ruiz-Usano [3]; Framinan [4]) and it also differs by employing two concatenations of forward and inverse reduction techniques and finally the local search involving an adjacent pair-wise interchange of machines. As for the generation of sequences of jobs for obtaining the supersequences of machines, we employ the PACO, the ant-colony algorithm proposed by Rajendran and Ziegler [10] for the permutation flowshop scheduling problem. This ant-colony algorithm is found to be one of the best algorithms for permutation flowshop scheduling (see Ruiz et al. [11] and Ruiz and Stuetzle [12]). It is to be noted that our proposed ant-colony algorithm constructs a sequence of jobs from which a feasible supersequence of machines is generated, whereas the ant-colony algorithm by Michel and Middendorf [9] and the genetic algorithm by Branke et al. [8] construct a full feasible supersequence of machines. Hence we find our ant-colony algorithm computationally simple; moreover, our string reduction techniques are also computationally simple in the sense that every reduced supersequence is checked for feasibility with respect to the machine routing of every job and this check is computationally straightforward and simple.

2 Proposed Ant-Colony Algorithm Integrated with String Formation and Reduction Techniques (PACO-SFR)

The salient features of the proposed ant-colony algorithm integrated with the novel features of string formation and reduction techniques (called PACO-SFR) are now presented, followed by a detailed discussion of the algorithm. The PACO-SFR first generates a sequence or string of jobs by following the procedure related to the generation of an ant-sequence described in the PACO (see Rajendran and Ziegler [10]) that uses the pheromone intensity or trail matrix $[\tau_{ij}]$ and hence a supersequence of machines is formed; on this supersequence or flowline of machines, it then employs two concatenations of forward and inverse reduction procedures, a machine reduction technique, and finally a local search involving an adjacent pair-wise interchange of machines in the flowline. The resultant flowline thus obtained and its length constitute the solution corresponding to the ant-sequence or string of jobs generated. The PACO-SFR employs the job-index based insertion scheme (called JIS) as a local search scheme to generate improved job sequences. The job sequence thus obtained with respect to the resultant supersequence of machines with the minimum string or flowline length is used to update the pheromone intensity or trail matrix $[\tau_{ij}]$. This algorithm is carried out over 40 times or iterations, and the best job sequence thus obtained (in terms of the minimum flowline length) is returned.

We discuss the complete algorithm with numerical illustrations through which we explain the mechanism of the PACO-SFR. The following jobshop problem with $n = 6$ jobs and with $m = 4$ machines, and with the following sequence of machine-routings is considered throughout in this paper (note that the sequence of visits of jobs on machines is as per the order given below):

job 1	: 1-2-3-4
job 2	: 1-3-2-4
job 3	: 1-4-2-3
job 4	: 2-3-1-4
job 5	: 2-4-1-3
job 6	: 3-4-1-2

2.1 Generation of a Supersequence or String of Machines

In our study we obtain a supersequence of machines by ordering jobs in a specific order and thereafter laying out machines in that order. For example, if we order the jobs in the order {1-2-3-4-5-6}, the corresponding initial supersequence of machines or flowline is {1-2-3-4-1-3-2-4-1-4-2-3-2-3-1-4-2-4-1-3-3-4-1-2} obtained by arranging machines for processing jobs in the given order {1-2-3-4-5-6}. This supersequence of machines is then reduced by employing the string reduction techniques presented in this study. Note that previous research attempts (e.g. those by Framinan and Ruiz-Usano [3]; Framinan, [4]) obtained the initial supersequence of machines or flowline {1-1-1-2-2-3-2-3-4-3-4-4-3-2-2-1-1-1-4-4-3-4-3-2} by

arranging the machines of the first operations of jobs in the given order {1-2-3-4-5-6}, then the machines of the second operations of jobs in that order, and so on up to the machines of the last operations of jobs taken in the given order {1-2-3-4-5-6}. However, from our computational experiments, we found that our arrangement of initial supersequence of machines has resulted in yielding flowlines with less lengths than those obtained from the arrangement of initial supersequence of machines suggested by previous researchers.

As an example, consider a two-job two-machine SCS problem given by Framinan [4]:

job 1	: 1-1-1-2-2-2
job 2	: 2-2-2-1-1-1

When we follow our approach of forming the initial supersequence of machines with machines laid out as per the job order {1-2}, we have {1-1-1-2-2-2-2-2-2-1-1-1} and after the application of the forward reduction technique or the inverse reduction technique (see their details in the text to follow), we have the resultant supersequence {1-1-1-2-2-2-1-1-1}. However, when the approach by Framinan [4] is followed, we have the initial supersequence {1-2-1-2-1-2-2-1-2-1-2-1}, and after the application of the forward reduction technique, we have the supersequence {1-2-1-2-1-2-1-2-1-2-1} which is longer than that obtained from our approach.

2.2 Implementation of the Set of String Reduction Techniques on a Job Sequence

The initial job sequence in our PACO-SFR is obtained by ordering jobs as {1-2-3-…-n} and hence we form a supersequence of machines. We then employ the string reduction techniques, namely, two concatenations of forward and inverse reduction techniques, machine elimination technique and a local search involving an adjacent pair-wise interchange of machines on the job sequence to get a reduced supersequence of machines. We first present the forward reduction procedure (see Framinan [4]). Considering one job at a time, we scan the given supersequence of machines from the left to see what machines are required to process the chosen job and mark the machines accordingly. After all jobs are considered, the marked machines survive in the supersequence of machines. For the supersequence of machines or flowline {1-2-3-4-1-3-2-4-1-4-2-3-2-3-1-4-2-4-1-3-3-4-1-2}, we get the reduced supersequence or flowline {1-2-3-4-1-3-2-4-3}, after this forward reduction of scanning from the left to the right of the given supersequence and satisfying the machine-routings with respect to every job.

As for the inverse reduction (also see Framinan [4]), considering one job at a time, we scan the supersequence of machines from the right to the left to see what machines are required to process the chosen job in the reverse sequence of operations and mark the machines accordingly. After all jobs are considered, the marked machines survive in the supersequence of machines. For the supersequence of machines {1-2-3-4-1-3-2-4-1-4-2-3-2-3-1-4-2-4-1-3-3-4-1-2}, we get the reduced flowline {1-2-3-1-4-2-4-1-3-4-1-2}.

It is evident that the forward reduction procedure need not yield the same resultant reduced supersequence of machines as that from the application of the inverse reduction. For this reason, Framinan [4] applied the forward reduction on the initial supersequence, and once again considering the initial supersequence, Framinan applied the inverse reduction procedure. The better of the two resultant supersequences is chosen by Framinan. However, according to our proposal, we explore two concatenations: apply first the forward reduction procedure on the initial supersequence and then apply the inverse reduction on the resultant reduced supersequence (called concatenation (forward + inverse)) to possibly reduce the supersequence of machines; apply first the inverse reduction procedure on the initial supersequence and then apply the forward reduction on the resultant reduced supersequence (called concatenation (inverse + forward)) to reduce the supersequence of machines. Then we take the better of these two reduced supersequences. For example, when we apply the concatenation (forward + inverse) on the supersequence {1-2-3-4-1-3-2-4-1-4-2-3-2-3-1-4-2-4-1-3-3-4-1-2}, we get the resultant reduced supersequence {1-2-3-4-1-3-2-4-3}; when we apply the concatenation (inverse + forward) on the supersequence {1-2-3-4-1-3-2-4-1-4-2-3-2-3-1-4-2-4-1-3-3-4-1-2}, we get the resultant reduced supersequence {1-2-3-1-4-2-4-1-3-2}. The better of these two supersequences is chosen by us for possible further reduction, i.e., {1-2-3-4-1-3-2-4-3}.

It is therefore evident that the two concatenations of forward reduction and inverse reduction serve to reduce the length of the supersequence of machines than the application of only one reduction technique. As a further example, we can show the effectiveness of these two concatenations with the same numerical illustration with the machine routing of job 4 changed to (2-1-4) (instead of (2-3-1-4)) and that of job 6 changed to (4-1-2) from (3-4-1-2). The concatenation (forward + inverse) yields the resultant supersequence of machines {1-2-4-1-3-2-4-3} (with the forward reduction first yielding {1-2-3-4-1-3-2-4-3} and the inverse reduction thereafter yielding {1-2-4-1-3-2-4-3}) and the concatenation (inverse + forward) yields {1-3-4-2-4-1-3-4-2} (with the backward reduction first yielding {1-3-1-4-2-4-1-3-4-1-2} and the forward reduction thereafter yielding {1-3-4-2-4-1-3-4-2}. The two resultant supersequences obtained by the concatenation of the forward and backward reduction techniques are of less length than those yielded by the single application of either the forward reduction technique or the inverse reduction technique. Reverting to our original example, we consider two supersequences {1-2-3-4-1-3-2-4-3} and {1-2-3-1-4-2-4-1-3-2} (obtained from two concatenations of forward and inverse reduction techniques), and choose the supersequence that has less length.

Let us see how we can reduce its string length further. Now we employ the machine elimination technique (also attempted by Framinan and Ruiz-Usano [3]) on the supersequence {1-2-3-4-1-3-2-4-3}, called *S*. We remove only one machine at a time from *S* and see if the resultant supersequence is feasible with respect to satisfying the machine routing of every job; if so, the last such reduced (by one element) supersequence is chosen. As an example, suppose we remove machine 1 (found first in the supersequence) and we have the supersequence {2-3-4-1-3-2-4-3}. This supersequence does not satisfy the machine routings of all jobs. Then we remove

machine 2 found in the second position of S with no success. However if we remove machine 3 found in position 6 in S, we have the resultant supersequence {1-2-3-4-1-2-4-3} and this supersequence satisfies all machine routings, thereby resulting in a reduced string length. We continue with this process of removing a machine from S until the last machine in S is considered for elimination. The last such reduced supersequence, if it exists, with the string length $|S|$-1 is chosen; otherwise S is retained. It is interesting to note that while the machine elimination technique serves to reduce the supersequence {1-2-3-4-1-3-2-4-3}, it does not reduce the supersequence {1-2-3-1-4-2-4-1-3-2}.

For the purpose of exploring a further reduction in string length, we employ a local search involving an adjacent interchange of machines in the supersequence, after the machine elimination technique. This local search works as follows. We swap the machines found in positions i and i+1, where i = 1, 2, …, $|S|$-1. Every such resultant supersequence is subjected to the concatenation (forward + inverse), and the best among the resultant feasible supersequences (feasible in terms of all jobs' machine routings being present in the supersequence) and S with the least string length is chosen. In the supersequence {1-2-3-1-4-2-4-1-3-2}, when we swap machines 1 and 4 found in adjacent positions, we have the resultant supersequence reduced to {1-2-3-4-1-2-4-3} and this is chosen because it is feasible with respect to machine routings of all jobs being present in it and it has a less string length than the original supersequence. This local search of adjacent pairwise interchange of machines is implemented twice successively to possibly reduce the string length to the extent possible.

Thus we obtain a supersequence of machines or flowline with possibly minimum string length obtained from the given sequence of jobs, and this string reduction is achieved through two concatenations of forward and inverse reduction techniques, followed by a machine elimination technique and a local search involving two-time application of the adjacent pair-wise interchange of machines in a supersequence.

2.3 Improvement of a Job Sequence Using the JIS

The effectiveness or performance of a job sequence is measured in terms of the length of the resultant supersequence of machines. The initial job sequence {1-2-….- n} is taken as the current seed sequence and subjected to the JIS three times successively for a possible improvement in its performance, and this sequence is taken as the seed sequence to the PACO-SFR to begin with. Let [k] denote the index of the job in position k of the current seed sequence of jobs and let i refer to the index of jobs.

Do the following:

for i = 1(1) n:
{
for k = 1(1) n:
{
if [k] ≠ i

then
insert job i in position k of the current seed sequence and adjust the sequence accordingly by not changing the relative positions of other jobs; determine the resultant supersequence of machines for this job sequence, apply the proposed set of string reduction techniques on the supersequence of jobs and hence compute the performance of the job sequence in terms of the length of the reduced supersequence of machines.
}
choose the best sequence among the generated (n-1) job sequences;
if the performance of this sequence (in terms of the length of the corresponding reduced supersequence of machines) is better than or equal to the performance of the current seed sequence, then the current seed sequence is replaced by the best sequence found above.
}

The current seed sequence finally returned by the three-time application of the JIS is in fact the possibly improved job sequence by the JIS in relation to the seed sequence to the JIS. This final sequence is the seed sequence to the PACO-SFR (called the best sequence of jobs as of now) and let the string length of this sequence be denoted by Z_{best}.

2.4 Initialization of Parameters in the PACO-SFR

We initialize the pheromone intensity or trail matrix as follows:

set $\tau_{ik} = (1/Z_{best})$,
if (|position of job i in the seed sequence to the PACO-SFR - k|+1) $\leq$ n/4;
$(1/(2\times Z_{best}))$,
if n/4 < (|position of job i in the seed sequence to the PACO-SFR - k|+1) $\leq$ n/2;
$(1/(4\times Z_{best}))$, otherwise.

The rationale behind this setting of τ_{ik}s is that the seed solution to the PACO-SFR being good, those positions that are close to the position of job i in the seed sequence should be associated with larger values of τ_{ik}s than those that are away from the position of job i in the seed sequence. ρ is set to 0.75 in our study.

2.5 Construction of an Ant Sequence and Its Improvement by the JIS

In order to build a complete ant sequence of jobs, the following procedure is used to choose an unscheduled job i for position k, starting from a null sequence, for k = 1, 2, …, n.

Set $T_{ik} = \sum_{q=1}^{k} \tau_{iq}$ and sample a uniform random number u in the range [0, 1].

If $u \leq 0.4$ then the first unscheduled job as present in the best sequence of jobs obtained so far is chosen;

else

if $u \leq 0.8$ then

among the set of the first five unscheduled jobs, as present in the best sequence of jobs obtained so far, choose the job with the maximum value of T_{ik} ;

else

job i is selected from the same set of five unscheduled jobs for position k as a result of sampling from the following probability distribution:

$$p_{ik} = \left(\frac{T_{ik}}{\sum_{l} T_{lk}} \right),$$

where job l belongs to the set of the first five unscheduled jobs, as present in the best sequence obtained so far (note that when there are less than five jobs unscheduled, then all such unscheduled jobs are considered).

A complete ant sequence of n jobs is constructed accordingly and thereafter the set of proposed string reduction techniques is applied for obtaining the reduced supersequence of machines corresponding to this ant sequence of jobs. This ant sequence of jobs is then subjected to the JIS three times and the final resultant sequence of jobs (called the current sequence) with the length of the corresponding reduced supersequence of machines denoted by $Z_{current}$. If this current sequence's $Z_{current}$ is same as or better than Z_{best}, then set this sequence and $Z_{current}$ as the best sequence and $Z_{current}$ respectively.

2.6 Updating of Pheromone Trails or Intensities

In the PACO-SFR, updating of the trail intensities is based not only on the resultant sequence of jobs obtained after the three-time application of the JIS on the ant sequence, but also on the relative distance between a given position and the position of job i in the resultant sequence, and also related to the best sequence obtained so far. The trails are updated as follows.

Let h be the position of job i in the resultant sequence;

set $(\tau_{ik})^{updated} = \rho \times (\tau_{ik})^{old} + (1/(diff \times Z_{current}))$, if $|h - k| \leq 1$;

$\rho \times (\tau_{ik})^{old}$, otherwise,

where $diff = (|\text{position of job } i \text{ in the best sequence obtained so far} - k| + 1)^{1/2}$. This differential setting is based on the premise that the jobs occupying positions in the current sequence closer to their respective positions in the best sequence obtained so far should get their corresponding trail intensities increased by larger values.

2.7 Termination Condition

The PACO-SFR is terminated after 40 iterations (i.e., after the generation of 40 ant-sequences followed by the three-time application of the JIS). We use the PACO-SFR to generate a total of about $123n^2$ job sequences.

3 Computational Evaluation of the PACO-SFR

We consider the seventy benchmark jobshop instances considered by Framinan [4] and execute our ant-colony algorithm to solve the jobshop transformation problem instances. Framinan had given the best string length achieved with respect to every problem instance as a result of the implementation of algorithms such as H2 and H3 due to Branke et al. [8], BS due to Framinan and Ruiz-Usano [3] and TS due to Framinan [4] (see Table 6 in Framinan [4]). It is to be noted that the length of the resultant flowshop for every jobshop problem instance, as reported by Framinan, is through a consolidation of all the mentioned algorithms and that Framinan had reported the CPU time requirements and not the number of transformations or job sequences enumerated in the process of obtaining the results reported in the paper before the final transformation was obtained from all algorithms considered. For the sake of standardizing the computational effort requirement independent of the computer, its operating system and the programming language, in our work we have noted the number of job sequences enumerated to get the best flowline-length so that future researchers would find it easy to relatively evaluate our work. Note that a job sequence leads to the generation of a supersequence of machines, followed by the application of string reduction techniques. The computational of string reduction techniques is quite small and is the same across all job sequences. We have also noted the final job sequence and the corresponding supersequence of machines obtained from the PACO-SFR for the sake of completely reporting our results for possible future reference for researchers, apart from noting the details regarding the supersequence of machines obtained from the initial job sequence improved by the three-time application of the JIS and the number of job sequences enumerated to obtain the best string length for every problem instance in the PACO-SFR. The results are presented in Table 1.

It is found that the proposed ant-colony algorithm yields better results than those reported by Framinan [4] for 32 jobshop problem instances (i.e., for 46% of the problem instances), the best known solutions for 32 problem instances (i.e., for 46% of the problem instances) and worse solutions only in six problem instances (i.e., in only 8% of the problem instances). It is also seen from the computational experiments that the initial supersequences of machines (obtained after the three-time application of the JIS on the job sequence {1-2-…-n}) are the same as the string lengths reported by the previous researchers in 25 problem instances, less in 8 problem instances than those reported so far and quite close in most problem instances, thereby demonstrating the effectiveness of the proposed string reduction techniques. The best solutions for the benchmark problem instances are shown in bold in Table 1.

4 Summary

In this work we have dealt with the problem of transforming jobshops into flowshops with the objective of minimizing the length of the flowshop. An ant-colony algorithm,

Table 1. Computational results

Jobshop problem instance	*n*	*m*	String length derived from the initial job sequence	Best string length obtained by the ant-colony algorithm	Best string length reported by Framinan [4]
La01	10	5	**13**	**13**	**13**
La02	10	5	**12**	**12**	**12**
La03	10	5	**12**	**12**	**12**
La04	10	5	**13**	**13**	**13**
La05	10	5	**12**	**12**	**12**
La06	15	5	**14**	**14**	**14**
La07	15	5	**14**	**14**	**14**
La08	15	5	**13**	**13**	**13**
La09	15	5	**14**	**14**	**14**
La10	15	5	**14**	**14**	**14**
La11	20	5	**14**	**14**	**14**
La12	20	5	**14**	**14**	**14**
La13	20	5	**15**	**15**	**15**
La14	20	5	**15**	**15**	**15**
La15	20	5	**14**	**14**	**14**
La16	10	10	35	**33**	35
La17	10	10	36	**34**	35
La18	10	10	37	**36**	38
La19	10	10	36	**31**	35
La20	10	10	36	**34**	35
La21	15	10	42	**39**	42
La22	15	10	44	**39**	41
La23	15	10	41	**40**	42
La24	15	10	42	**40**	43
La25	15	10	43	**41**	42
La26	20	10	47	44	**43**
La27	20	10	**44**	**44**	**44**
La28	20	10	49	**45**	46
La29	20	10	49	**45**	**45**
La30	20	10	47	**44**	45
La31	30	10	51	**49**	**49**
La32	30	10	51	**49**	**49**
La33	30	10	53	**49**	**49**
La34	30	10	52	50	**49**
La35	30	10	53	50	**49**
La36	15	15	83	**78**	79
La37	15	15	80	**74**	79
La38	15	15	83	**77**	80
La39	15	15	79	**75**	80
La40	15	15	81	**75**	78
Orb01	10	10	29	**27**	28
Orb02	10	10	33	**32**	34
Orb03	10	10	**20**	**20**	**20**
Orb04	10	10	34	**33**	34
Orb05	10	10	28	**26**	**26**

Table 1. *(continued)*

Orb06	10	10	29	**27**	28
Orb07	10	10	33	**32**	34
Orb08	10	10	**20**	**20**	**20**
Orb09	10	10	34	**33**	34
Orb10	10	10	28	**26**	**26**
swv01	20	10	**29**	**29**	**29**
swv02	20	10	29	**27**	28
swv03	20	10	28	**27**	28
swv04	20	10	30	**29**	**29**
swv05	20	10	**29**	**29**	30
swv06	20	15	62	**59**	61
swv07	20	15	63	**57**	59
swv08	20	15	59	**56**	60
swv09	20	15	62	**56**	59
swv10	20	15	60	**57**	**57**
swv11	50	10	34	**32**	**32**
swv12	50	10	34	**33**	**33**
swv13	50	10	34	**32**	**32**
swv14	50	10	33	**32**	33
swv15	50	10	34	**33**	**33**
swv16	50	10	59	54	**52**
swv17	50	10	58	**53**	54
swv18	50	10	58	56	**54**
swv19	50	10	59	55	**53**
swv20	50	10	58	**55**	**55**

called PACO-SFR, is proposed with the integration of string reduction techniques in the ant-colony algorithm. The performance of the ant-colony algorithm is relatively evaluated by considering the best reported work and with the consideration of benchmark jobshop problem instances. It is found that the proposed ant-colony algorithm obtains better solutions and the best known solutions in most problem instances.

Acknowledgments. The first author gratefully acknowledges the support from DAAD for her research stay in the University of Passau during May-July, 2010. The second author thanks Alexander von Humboldt Stiftung for the financial support and C R Chandrasekaran for his initial involvement in this work. The authors thank the three reviewers for their comments and suggestions to improve the earlier version of the paper.

References

1. Knolmayer, G., Mertens, P., Zeier, A.: Supply Chain Management based on SAP Systems. Springer, Berlin (2002)
2. Kimms, A.: Minimal investment budgets for flow line configuration. IIE Transactions 32, 287–298 (2000)

3. Framinan, J.M., Ruiz-Usano, R.: On transforming job-shops into flow-shops. Production Planning and Control 13, 166–174 (2002)
4. Framinan, J.M.: Efficient heuristic approaches to transform job shops into flow shops. IIE Transactions 37, 441–451 (2005)
5. Raiha, K.J., Ukkonen, E.: The shortest common supersequence problem over binary alphabet is NP-complete. Theoretical Computer Science 16, 187–198 (1981)
6. Timkovsky, V.G.: Complexity of common subsequences and supersequences problems and related problems. Cybernetics 25, 565–580 (1990)
7. Framinan, J.M.: An adaptive branch and bound approach for transforming job shops into flow shops. Computers & Industrial Engineering 52, 1–10 (2007)
8. Branke, J., Middendorf, M., Schneider, F.: Improved heuristics and a genetic algorithm for finding short supersequences. OR Spektrum 20, 39–46 (1998)
9. Michel, R., Middendorf, M.: An ACO algorithm for the shortest common supersequence problem. In: New Ideas in Optimization. McGraw-Hill, Maidenhead (1999)
10. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155, 426–438 (2004)
11. Ruiz, R., Maroto, C., Alcaraz, J.: Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34, 461–476 (2006)
12. Ruiz, R., Stuetzle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operations Research 177, 2033–2049 (2007)