Journal of the Operational Research Society (2005) 56, 947–953 © 2005 Operational Research Society Ltd. All rights reserved. 0160-5682/05 $30.00

www.palgrave-journals.com/jors

# A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs

O Holthaus[1]* and C Rajendran[2]

[1]*University of Passau, Passau, Germany; and* [2]*Indian Institute of Technology Madras, Chennai, India*

The problem of scheduling on a single machine is considered in this paper with the objective of minimizing the sum of weighted tardiness of jobs. A new ant-colony optimization (ACO) algorithm, called fast ACO (FACO), is proposed and analysed for solving the single-machine scheduling problem. By considering the benchmark problems available in the literature for analysing the performance of algorithms for scheduling on a single machine with the consideration of weighted tardiness of jobs, we validate the appropriateness of the proposed local-search schemes and parameter settings used in the FACO. We also present a comparison of the requirements of CPU time for solving the single-machine total-weighted tardiness problem by the FACO and the existing algorithms.

*Journal of the Operational Research Society* (2005) **56**, 947–953. doi:10.1057/palgrave.jors.2601906
Published online 22 December 2004

**Keywords:** single machine; scheduling; total-weighted tardiness; ant-colony algorithm

## Introduction

The problem of scheduling has been keenly researched over the last several years. The problem of scheduling jobs on a single machine (or machines in series) is NP-hard in most cases; for example, the problem of scheduling jobs on a single-machine with the objective of minimizing the sum of tardiness of jobs (or with the objective of minimizing the sum of weighted tardiness of jobs) is NP-hard,[1,2] and attempts have been made to use enumerative algorithms to solve the problem to optimality.[3] Many heuristics have been proposed to solve the problem of scheduling of jobs on a single-machine in order to obtain a near-optimal solution in a polynomial time.[4,5] The most effective heuristics, till date, appear to be the iterated dynasearch algorithm by Congram *et al*,[6] and the iterated local-search algorithm by den Besten *et al*.[7] The salient feature of the dynasearch algorithm is that it employs dynamic programming (to find the best combination of moves) in a local-search algorithm and it allows a series of moves at each iteration to be performed with look-ahead capability to possibly prevent the search getting trapped in local minima. The iterated local-search algorithm by den Besten *et al* aims at configuring an iterated local search so as to optimize the interaction of single-procedure parts of the local search through the consideration of variable neighbourhood descent, perturbation strength, nature of perturbations, and an acceptance criterion.

In recent times, ant-colony-optimization algorithms (or simply, ant-colony or ACO algorithms) are increasingly used to solve combinatorial optimization problems. An introduction to the ACO algorithms has been dealt with in Dorigo *et al*.[8] Attempts have been made recently to solve the single-machine scheduling problems by using ACO algorithms.[9]

In this paper, we investigate the problem of scheduling on a single machine with the objective of minimizing the sum of weighted tardiness of jobs, and propose an ant-colony optimization algorithm, called the FACO. The proposed ant-colony algorithm is executed so as to attain either the optimal or the best-known value of the sum of weighted tardiness of jobs for a given problem, and the CPU time is noted. We compare this CPU time with the CPU times required by the existing algorithms. We also present the performance analyses with respect to various local-search schemes and parameter settings involved in the FACO so as to justify the prescribed schemes and settings.

## Description of the problem and the proposed fast ACO algorithm (FACO)

### *Description of the problem under study*

We consider the problem of static scheduling of jobs on a single-machine. Suppose there are $n$ jobs available at time zero, with $t_i$ denoting the process time of job $i$ on the machine. Let $D_i$ denote the due-date of job $i$ and $w_i$ denote the weight (or relative tardiness cost or relative importance)

*Correspondence: O Holthaus, Department of Production, Operations and Logistics Management, Faculty of Business Administration and Economics, University of Passau, Innstrasse 39, Bavaria, 94032 Passau, Germany.*
E-mail: Oliver.Holthaus@Uni-Passau.De

for job $i$. Suppose $C_i$ denotes the completion time of job $i$ in a given sequence, say, $\Omega$. We have the sum of weighted tardiness of jobs, $Z(\Omega)$, in $\Omega$ given as follows:

$$Z(\Omega) = \sum_{i=1}^{n} \max\{C_i - D_i;\ 0\} \times w_i \qquad (1)$$

### General structure of ant-colony algorithms

The principle in ant-colony algorithms is based on the use of pheromone trails laid by real ants as a medium for communication and feedback among ants. ACO algorithms make use of simple agents called *ants* that iteratively construct solutions to combinatorial optimization problems. The solution generation or construction by ants is guided by (artificial) pheromone trails based on the value of the objective function. An individual ant constructs a complete solution by starting with a null solution and iteratively adding solution components until a complete solution is constructed. The trails form a kind of adaptive memory of previously found solutions and are modified at the end of each iteration. In the context of application of ACO algorithms to scheduling problems, $\tau_{ik}$ denotes the trail intensity (or desire) of placing job $i$ in position $k$ of a sequence.

The general structure of the proposed ACO algorithm can be described as follows:

*Step 1*: Initialize the pheromone trails and parameters.
*Step 2*: While (termination condition is not met) do the following:
- construct a solution;
- improve the solution by local-search techniques;
- update the pheromone trails or trail intensities.

*Step 3*: Return the best solution found.

### Details of the proposed Fast ACO algorithm

We now present the details of the proposed ant-colony algorithm called the FACO in the following steps.

### Obtaining the seed solution for initializing the trail intensities

The seed sequence for the ant-colony algorithm is obtained as follows: first, we generate a sequence of jobs by ordering them in the ascending order of their due-dates (called the EDD sequence). The sequence thus obtained is taken as the sequence to be improved upon by the application of the well-known improvement scheme, called NEH improvement scheme[10] (see Appendix for details of the adaptation of the scheme to the problem under study). We have chosen to use the EDD sequence as the seed sequence because a related work on ant-colony algorithm[9] also investigated the use of EDD sequence, to start with.

In order to further improve the quality of the solution (ie the sequence yielded by the improvement scheme of Nawaz *et al*), we have implemented on this sequence two proposed improvement schemes, called Random-Job-Insertion Scheme (RINS) and Random-Job-Pair-Swap Scheme (RJPS) (see Appendix for details).

After subjecting the sequence (yielded by the NEH improvement scheme) to RINS once and then to RJPS twice (and this concatenated application of RINS and RJPS being carried out twice), the resultant sequence, namely, $\Omega$, is set to $\Omega^*$, where $\Omega^*$ denotes the best sequence obtained so far in the FACO. Let the sum of weighted tardiness of jobs in $\Omega^*$ be denoted by $Z_{\text{best}}$. The concatenated application of these two local-search schemes is done in view of the observation that a local optimum with respect to one neighbourhood structure need not be a local optimum for the other one. Sequence $\Omega$ is taken as the input to the FACO, to start with. Note that it is possible that $Z_{\text{best}}$ is equal to zero, in which case we do not execute the FACO as the optimal solution is already reached.

### Initialization of trail intensities

We initialize the trail intensities as follows:

```
For $i = 1$ to $n$ do /* set the trail intensities for all jobs */
    For $k = 1$ to $n$ do
        /* set the trail intensities for a given job with respect
        to all positions in the seed sequence */
        set $\tau_{ik} = (1/(Z_{\text{best}} \times \sqrt{dist}))$
```

$$\text{where } dist = (|\text{position of job } i \text{ in } \Omega - k| + 1) \qquad (2)$$

The rationale behind this initial differential setting of the $\tau_{ik}$'s is that the seed solution to the FACO being good, those positions that are close to the position of job $i$ in the seed sequence should be associated with larger values of the $\tau_{ik}$'s than those positions that are away from the position of job $i$ in the seed sequence. In other words, the influence of a good seed sequence is better reflected in such a differential setting of the $\tau_{ik}$'s, as opposed to the same setting of the $\tau_{ik}$'s, with respect to position $k$ for job $i$.

### Construction of an ant-sequence and its improvement by the application of the RINS and RJPS

In the FACO, the following procedure is used to choose an unscheduled job $i$ for position $k$.

```
Set $T_{ik} = \sum_{q=1}^{k} \tau_{iq}$ and sample a uniform random number
$u$ in the range [0, 1].
If $u \leqslant 0.4$
    then
        the first unscheduled job as present in $\Omega^*$ is chosen;
    else
        if $u \leqslant 0.8$, /* i.e., when $0.4 < u \leqslant 0.8$ */
            then
    among the set of the first five unscheduled jobs,
    as present in $\Omega^*$, choose the job with the
    maximum value of $T_{ik}$;
else/* i.e., when $u > 0.8$ */
    job $i$ is selected from the same set of five unsche-
    duled jobs for position $k$ as a result of sampling
    from the following probability distribution:
```

$$p_{ik} = \left( T_{ik} / \sum_{l} T_{lk} \right) \quad (3)$$

```
    where job $l$ belongs to the set of first five
    unscheduled jobs, as present in $\Omega^*$.
    /* note that when there are less than five jobs
    unscheduled, then all such unscheduled jobs are
    considered. */
```

A complete ant-sequence is thus generated by considering the unscheduled jobs and performing the above procedure for $k = 1, 2, \ldots, n$.

The rationale behind the selection of the job to be scheduled next is that the choice is determined with equal probability of choosing on the basis of the best sequence and the best value of $T_{ik}$ (ie with the probability being 0.4) and the probabilistic choice of the job is made with a probability of 0.2 which is half of the probability of going in for the first unscheduled job found in the best sequence. Moreover, the experimentation with different probability ranges has shown the proposed settings in Expression (3) to perform the best.

The generated ant-sequence is subjected to the successive application of the RINS once and the RJPS twice, with this concatenation being carried out twice. Let the objective-function value of the resultant sequence be denoted by $Z_{\text{current}}$.

### *Updating of trail intensities*

In the FACO, the trail intensities are updated as follows.

```
Let $h(i)$ be the position of job $i$ in the resultant sequence.
For $i = 1$ to $n$ do
    For $k = 1$ to $n$ do
        If $h(i) = k$
        then
            set $\tau_{ik} := \rho \times \tau_{ik} + (1/Z_{\text{current}})$
        else
            set $\tau_{ik} := \rho \times \tau_{ik}$                    (4)
```

where $\rho$ denotes the persistence of trail (set to 0.80 in this study after pilot runs).

If $Z_{\text{current}}$ is better than $Z_{\text{best}}$, then $\Omega^*$ and $Z_{\text{best}}$ are updated accordingly.

### *Generation of ant-sequences and termination of the FACO*

In the FACO, we keep on generating ant-sequences (with every generated ant-sequence being subjected to the two-time concatenated application of the RINS once and the RJPS twice) until we find that there has been no improvement in $Z_{\text{best}}$ in the last 30 iterations involving the construction of an ant-sequence and its improvement, and updating of trail intensities. When we find no improvement in such a case, we generate a sequence at random, set it to $\Omega$, implement the two-time concatenated application of the RINS once and the RJPS twice on $\Omega$, and compare the value of objective function yielded by $\Omega$ with $Z_{\text{best}}$. Update $\Omega^*$ and $Z_{\text{best}}$, if necessary. Start executing the three main steps initialization of trail intensities, construction of an ant-sequence and its improvement, and updating of trail intensities, until we find no improvement in $Z_{\text{best}}$ in the last 30 generated (and subsequently improved through local-search schemes) ant-sequences. Basically, we resort to multi-start approach in the FACO. However, the best sequence obtained so far in the search is always kept track of, and is used as a basis for the construction of an ant-sequence (see Expression (3)). This procedure is repeated until the optimal or best-known value of the objective function is reached for a given problem instance. The corresponding CPU time is noted. The reason for resorting to this mode of termination is due to the fact that the existing algorithms such as the one by den Besten *et al*[9] also terminate whenever the optimal solution or the best-known solution for a given problem (see OR-Library at http://mscmga.ms.ic.ac.uk/jeb/orlib/wtinfo.html) is reached. In order to have a proper comparison with the existing methods, we have set this termination criterion.

The proposed FACO differs from the existing ant-colony algorithms in many aspects. For example, when we compare with the ant-colony algorithm of den Besten *et al*,[9] we find that the FACO does not use any heuristic desirability of assigning job $i$ to position $k$ (denoted by $\eta_{ik}$ in their paper), and that the initialization of trail intensities (as given in Expression (2) in this paper) is unique in the sense that the heuristic desirability is captured in a different way, and that the heuristic information is used only for initializing the trail intensities and not for updating them. Moreover, the rationale of choosing the job to be appended next is based on the use of the best-solution obtained so far and the use of $T_{ik}$ (see Expression (3)); in addition, the local-search schemes, namely, the RINS and RJPS, are new and different, apart from the introduction of the multi-start approach in the FACO.

It is to be noted that we have undertaken a sensitivity analysis of performance of the FACO by varying the settings of different parameters such as those involved in the choice of the job to be appended next, and the updating of trail intensities. As for the single-machine scheduling problem under study, we have found that the prescribed settings have resulted in the best performance of the FACO. Some important relevant details are reported in the next section.

## Performance analysis of the FACO

Since the ant-colony algorithm is a metaheuristic that involves sampling, we have replicated the execution of the FACO 25 times, and observed the mean, maximum, minimum and standard deviation of CPU times over 25 replications. This is performed so to report the performance of the FACO for different random-number streams and hence gauge the correct performance of the FACO in a better manner. Moreover, other researchers have also used 25 replications or runs in solving the single-machine total-weighted-tardiness problem.[9] We have made use of the benchmark problems with the consideration of 40, 50 and 100 jobs (taken from the OR-Library available at http://mscmga.ms.ic.ac.uk/jeb/orlib/wtinfo.html), and solved every problem by using the FACO either to optimality or the best-known value of total-weighted tardiness of jobs (except in the phase of the study involving the evaluation of the FACO and iterated multi-start algorithm, called M-S, where we have executed the algorithms not exceeding 5 s of CPU time). We have used an 800-MHz Pentium PC and C++ under WIN NT 4.0 for execution of the FACO and the related variants to follow.

Before we present a comparative analysis of the FACO and the existing algorithms for solving the single-machine weighted-tardiness problem, we present the details of performance analysis of various local-search schemes and parameter settings involved in the FACO.

First, we have taken up the investigation of the effectiveness of various local-search schemes used in this paper, namely, the use of NEH scheme as an intermittent improvement or local-search scheme, and the use of the RINS and RJPS. In this phase of experimental investigation, we do not incorporate the ant-colony algorithm; we rather investigate only the relative effectiveness of local-search schemes without the use of ant-colony algorithm. The following variants of local-search schemes are evaluated.

1. *Variant 1*: The generation of seed sequence by using the EDD rule, followed by the two-time implementation of the RINS once and the RJPS twice. It means that the sequence obtained by the use of the EDD rule is not subjected to the NEH improvement scheme before the use of the RINS and RJPS.
2. *Variant 2*: The generation of the seed sequence by using the EDD rule, followed by the implementation of NEH improvement scheme, and the two-time concatenated implementation of the RINS once and the RJPS twice.
3. *Variant 3*: The generation of the seed sequence by using the EDD rule, followed by the NEH improvement scheme, and four-time application of the RINS.

In each of the above variants, we have performed 25 replications or runs. The purpose of testing Variants 1 and 2 is to test for the relative effectiveness of the use of the NEH improvement scheme as an intermediate local search scheme. The purpose of Variants 2 and 3 is to test for the relative effectiveness of the concatenation of the RINS and RJPS, as opposed to the use of a single local search scheme.

In order to evaluate the variants, we have considered the 100-job problem instances, and made use of the following performance measures:

1. the average percentage relative deviation of the solution yielded by a variant, in comparison with the optimal or the best-known value for the given problem instance (obtained by computing 'solution yielded by a variant × 100/the optimal or the best-known value for the given problem instance', and the average being computed from such percentage relative deviations over 25 replications and 125 problem instances with 100 jobs, and denoted by $PD_{avg}$); note that the percentage relative deviation has not been computed and taken into account while computing the $PD_{avg}$, if the sum of weighted tardiness of jobs is equal to zero;
2. the maximum percentage relative deviation of the solution yielded by a variant, in comparison with the optimal or the best-known value for the given problem instance, the maximum value being computed from the percentage relative deviations among 25 replications and 125 problem instances with 100 jobs (denoted by $PD_{max}$);
3. the average absolute deviation of the solution yielded by a variant from the optimal or the best-known value for the given problem instance (the average being computed over the absolute deviations over 25 replications and 125 problem instances with 100 jobs) (denoted by $AD_{avg}$);
4. the maximum absolute deviation of the solution yielded by a variant from the optimal or the best-known value for the given problem instance, the maximum value being computed from the absolute deviations among 25 replications and 125 problem instances with 100 jobs (denoted by $AD_{max}$);
5. the average number of problem instances for which the optimal or the best-known value is obtained, the average being again computed over 25 replications and 125 problem instances (denoted by $\#opt_{avg}$); and
6. the average CPU time in seconds (denoted by $t_{avg}$).

The results are presented in Table 1.

It is evident from the table that the use of the NEH scheme as an intermittent improvement scheme proves quite effective, the scheme being well known in the development of heuristics in flowshop scheduling.[11–13] The effectiveness of

**Table 1** The performance of different variants involving local search schemes investigated in the paper

| *Variant* | $PD_{avg}$ | $PD_{max}$ | $AD_{avg}$ | $AD_{max}$ | $\#opt_{avg}$ | $t_{avg}$ |
|---|---|---|---|---|---|---|
| 1 | 0.933 | 98.2 | 276.5 | 4233 | 41.28 | 0.0109 |
| 2 | 0.220 | 15.0 | 148.7 | 2294 | 50.20 | 0.0164 |
| 3 | 1.581 | 36.4 | 808.3 | 15522 | 32.12 | 0.0164 |

concatenation of the local-search schemes, namely, that of the RINS and the RJPS, is due to the fact that each of these schemes perturbs the seed sequence in different ways, thereby discovering more local minima in the neighbourhood than a single local-search scheme. Hence Variant 2 of local-search schemes is chosen for incorporation in the FACO. It is to be noted that while we have experimented with various such alternate local-search schemes and concatenations, all such details are not given in this paper for the sake of conciseness, and that only the most relevant and important experimental aspects have been discussed.

Having justified the choice of local-search schemes, our next task is to prove the effectiveness of the basic ant-colony algorithm itself. In other words, we now investigate the effectiveness of the FACO and the effectiveness of an iterated multi-start local-search algorithm in which we use several randomly generated sequences, each followed by the implementation of the NEH improvement scheme, and the two-time application of the RINS and RJPS. It means that the multi-start local-search algorithm does not incorporate the ant-colony algorithm and hence the generation of ant-sequences. We call this algorithm M-S algorithm. Of course, we use the seed sequence generated by the EDD rule as one of the seed sequences in the M-S algorithm. Further, while we have executed the FACO with a time restriction of 5 s per run per problem instance, we have executed the M-S algorithm with a time restriction of 10 s per run per problem instance. It means that the M-S algorithm has been executed with 'greater advantage'! It has been observed that in most cases, the optimal or the best-known solutions have been obtained even before the respective upper limits of CPU time are reached. The performance measures are the same as the ones used in the case of earlier variants. The results are given in Table 2. It is clear from the table that the FACO outperforms the M-S algorithm, thereby proving the effectiveness of the proposed ant-colony algorithm. As another attempt to improve the performance of the FACO, we have obtained the seed sequence to the NEH improvement scheme in the FACO by arranging the jobs in the non-decreasing order of the value of process time divided by the weight for job-tardiness, the rule being called the Weighted-Shortest-Process-Time rule (WSPT rule), instead of arranging jobs by using the EDD rule. We term this variant of FACO as FACO$^{\mathrm{WSPT}}$. The computational experience with this variant is given in Table 2. It appears that the FACO$^{\mathrm{WSPT}}$ performs better than the FACO (with the EDD-based seed sequence). The reason is that when due-dates of jobs are tight, the WSPT-based sequence performs very well in terms of reducing the sum of weighted tardiness of jobs. Nevertheless, we also report the performance of the FACO with the EDD-based seed sequence because the EDD rule is often used as a seed sequence in related research attempts.[9]

Having established the appropriate use of different local-search techniques and the basic ant-colony algorithm, our next task is to justify the appropriateness of different parameter values used in the ant-colony algorithm proposed in this paper. For this purpose, we impose no restriction on the upper limit on CPU time and let every variant of the FACO run until the optimal or the best-known solution is obtained for every problem instance for every replication. This is carried out to primarily compare with the CPU times required by the existing algorithms as well because the existing algorithms were also executed until the optimal solution or the best-known solution had been reached. Although we have evaluated many such variants of the FACO, we present the results with respect to some important variants of the FACO, with the following prescribed parameter settings:

| | |
|---|---|
| FACO-1: | The FACO with *dist* set to 1 (see Expression (2)) —it means that all $\tau_{ik}$'s are set to the same value initially; |
| FACO-2: | The FACO with $\tau_{ik}$'s being used, instead of $T_{ik}$'s (see construction of an ant-sequence); and |
| FACO-3: | The FACO with *dist* set to 1 and the use of $\tau_{ik}$'s, instead of $T_{ik}$'s. |

The measures of performance are the average CPU time ($t_{\mathrm{avg}}$), standard deviation of CPU time ($t_{\sigma}$), the minimum CPU time ($t_{\mathrm{min}}$), and the maximum CPU time ($t_{\mathrm{max}}$), computed over 125 problem instances with 100 jobs. The results are presented in Table 3. It is evident from the table that the FACO and the FACO$^{\mathrm{WSPT}}$ require, on an average, the minimum computational effort for solving the problem.

Our final task is to present the details of the CPU time requirement, by considering the 100-job 125 problem instances, for the FACO, the FACO$^{\mathrm{WSPT}}$, the existing ant-colony algorithm by den Besten *et al*,[9] the iterated local

**Table 2** The performance of FACO, FACO$^{\mathrm{WSPT}}$ and multi-start algorithms with restricted execution time of 5, 5 and 10 s, respectively

| *Algorithm* | $PD_{avg}$ | $PD_{max}$ | $AD_{avg}$ | $AD_{max}$ | $\#opt_{avg}$ | $t_{avg}$ |
|---|---|---|---|---|---|---|
| FACO | 0.00006 | 0.028 | 0.04 | 16 | 124.4 | 0.222 |
| M-S | 0.02556 | 6.642 | 1.48 | 422 | 119.8 | 1.076 |
| FACO$^{\mathrm{WSPT}}$ | 0.00004 | 0.028 | 0.02 | 16 | 124.7 | 0.182 |

**Table 3** The performance of different variants of the FACO in terms of CPU time

| *Algorithm* | $t_{avg}$ | $t_{\sigma}$ | $t_{min}$ | $t_{max}$ |
|---|---|---|---|---|
| FACO | 0.2451 | 0.7957 | 0.0044 | 5.6986 |
| FACO-1 | 0.3641 | 0.9998 | 0.0044 | 5.0977 |
| FACO-2 | 0.3601 | 0.8537 | 0.0044 | 5.6905 |
| FACO-3 | 0.5074 | 1.3198 | 0.0044 | 6.8583 |
| FACO$^{\mathrm{WSPT}}$ | 0.2519 | 0.7964 | 0.0085 | 5.3672 |

**Table 4** Computational requirements of various algorithms (in seconds) computed over 125 problem instances with 100 jobs

| *Algorithm* | $t_{avg}$ | $t_{\sigma}$ | $t_{min}$ | $t_{max}$ | *Machine details* |
|---|---|---|---|---|---|
| FACO | 0.2451 | 0.7957 | 0.0044 | 5.6986 | 800 MHz Pentium-III PC with 512 MB RAM |
| FACO$^{WSPT}$ | 0.2519 | 0.7964 | 0.0085 | 5.3672 | 800 MHz Pentium-III PC with 512 MB RAM |
| den Besten *et al*[9] | 6.9900 | 7.0190 | 0.0180 | 86.2600 | 450 MHz Pentium-III PC with 256 MB RAM |
| den Besten *et al*[7] | 5.7500 | 14.5000 | 0.0076 | 105.5000 | 700 MHz Pentium-III PC with 512 MB RAM |
| Congram *et al*[6] | 36.4000 | Not reported | Not reported | Not reported | HP 9000-G50 with about 100 MHz |

Note: Congram *et al* were able to solve 123.2 problems, on average, either to optimality or to the best-known values of total-weighted tardiness of jobs, while others were able to solve all problems either to optimality or to the best-known values.

search algorithm by den Besten *et al*,[7] and the dynasearch algorithm by Congram *et al*.[6] It is to be noted that these algorithms have been executed by different authors on different machines with different speeds (in terms of MHz.); however, we present those details for the sake of having an idea about the computational time requirements by different algorithms, as reported in their respective papers. The results are presented in Table 4. It appears that the proposed FACO and FACO$^{WSPT}$ are computationally quite efficient and possibly faster than the existing algorithms.

## Summary

The problem of scheduling on a single-machine with the objective of minimizing the sum of weighted tardiness of jobs has been considered in this paper. A fast ant-colony algorithm has been proposed. A number of computational experiments with benchmark problems have been conducted to validate the appropriateness of local-search schemes and parameter settings used in the proposed ant-colony algorithm. It also appears that the proposed ant-colony algorithm is computationally attractive in comparison with the existing algorithms for solving the single-machine weighted-tardiness problem.

*Acknowledgements*—The second author gratefully acknowledges the Research Fellowship of the Alexander-von-Humboldt Foundation for carrying out this work in 2002, 2003 and 2004. The authors thank the referee for the suggestions and comments that improved the earlier version of the paper.

## References

1 Lawler EL (1977). A 'pseudopolynominal' algorithm for sequencing jobs to minimize total tardiness. *Ann Discrete Math* **1**: 331–342.

2 Du J and Leung JY-T (1990). Minimising the total tardiness on one machine is NP-hard. *Math Ops Res* **15**: 483–495.

3 Abdul-Razaq TS, Potts CN and Van Wassenhove LN (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete App Math* **26**: 235–253.

4 Potts CN and Van Wassenhove LN (1991). Single machine tardiness sequencing heuristics. *IIE Trans* **23**: 346–354.

5 Crauwels HAJ, Potts CN and Van Wassenhove LN (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS J Compu* **10**: 341–350.

6 Congram RK, Potts CN and van de Velde SL (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS J Compu* **14**: 52–67.

7 den Besten M, Stuetzle T and Dorigo M (2001). Design of iterated local search algorithms: an example application to the single machine total weighted tardiness problem. In: Boers EJW *et al* (eds). *Proceedings of EvoWorkshop 2001, Lecture Notes in Computer Science*, Vol. 2037 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, pp 441–451.

8 Dorigo M, Maniezzo V and Colorni A (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Trans Systems, Man Cybernetics—Part B* **26**: 29–41.

9 den Besten M, Stuetzle T and Dorigo M (2000). Ant colony optimizatioin for the total weighted tardiness problem. In: Schoenauer M *et al* (eds). *Proceedings of PPSN-VI*, Vol. 1971 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, pp 611–620.

10 Nawaz M, Enscore Jr EE and Ham I (1983). A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA* **11**: 91–95.

11 Rajendran C (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *Int J Prod Econ* **29**: 65–73.

12 Ho JC (1995). Flowshop sequenceing with mean flowtime objective. *Eur J Opl Res* **81**: 571–578.

13 Framinan JM, Leisten R and Rajendran C (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static flowshop sequencing problem. *Int J Prod Res* **41**: 121–148.

## Appendix

### *Improvement scheme used by Nawaz et al*:[10]

Suppose we have five jobs to be sequenced. In the case of the general *m*-machine flowshop problem, the jobs are arranged in the descending order of their total process times on all machines. In our problem, the jobs are arranged in the non-decreasing order of their due-dates. Let that ordered set be {5-4-3-2-1}. Take the first job from this set, that is, job 5, and hence form a partial sequence {5} with one job in it. Take the second job from the ordered set, that is, job 4, and insert it in all possible positions of the existing partial sequence, that is, {5}. Hence we get partial sequences {4-5} and {5-4}. Evaluate these partial sequences with respect to the objective function under consideration, that is, the sum of weighted tardiness of jobs. Choose the better of the two partial sequences; say, partial sequence {4-5}. Now take the job found next in the ordered set, that is, job 3 and insert it in all possible positions of partial sequence {4-5} to get new partial sequences {3-4-5}, {4-3-5} and {4-5-3}. Choose the best partial sequence among these new partial sequences with respect to the sum of weighted tardiness of jobs. The procedure progresses thus until a complete sequence of jobs is built up.

### *Step-by-step procedure of the RINS*

RINS performs the following steps once.

*Step 1*: Call the input sequence to the RINS $\Omega$. Let $[k]$ denote the job found in position $k$ of $\Omega$. Generate a sequence of $n$ jobs at random. Call it $\Psi$.

*Step 2*: Let the first job in $\Psi$ be $i'$.

*Step 3*: For $k=1$ to $n$, do the following:
if $i' \neq [k]$
then insert job $i'$ in position $k$ of $\Omega$ and adjust the sequence accordingly by not changing the relative positions of other jobs in $\Omega$; calculate the value of objective function of the resultant sequence.

*Step 4*: Choose the best sequence among the $(n-1)$ sequences generated in Step 3. If the value of objective function is improved, replace $\Omega$ by the best generated sequence.

*Step 5*: Remove the first job, that is, job $i'$, from $\Psi$. If $\Psi$ is not a null set, go back to Step 2; else stop. Sequence $\Omega$ is the output sequence from the RINS.

The RINS generates a total of $n(n-1)$ sequences in the search process.

### *Step-by-step procedure of the RJPS*

The RJPS works as follows. There are $(n(n-1))/2$ possible pairs of jobs that can be formed in a set of $n$ jobs. For example, we have {1, 2}, {1, 3},...{1, $n$}, {2, 3}, {2, 4},..., {2, $n$}, {3, 4},..., and {$n-1$, $n$} pairs of jobs as the total number of pairs. Choose a job-pair at random and swap the corresponding jobs in $\Omega$, the seed sequence to the RJPS, without altering the positions of other jobs. If the resultant sequence is better than $\Omega$ (in terms of the value of objective function), replace $\Omega$ with the generated sequence; else retain the seed sequence. Omitting this pair from further consideration, choose a job-pair at random and swap the jobs. Compare the generated sequence and $\Omega$, and update $\Omega$, if necessary. Proceeding likewise, we obtain the final sequence yielded by the RJPS, denoted by $\Omega$.

The RJPS enumerates a total of $(n(n-1))/2$ sequences in the search process.

*Received August 2003;*
*accepted August 2004 after one revision*