ELSEVIER

European Journal of Operational Research 142 (2002) 250–257

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

An Ant Colony Optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem

Vincent T'kindt *, Nicolas Monmarché, Fabrice Tercinet, Daniel Laügt

Laboratoire d'Informatique, Ecole d'Ingénieurs en Informatique pour l'Industrie, 64 avenue Jean Portalis, 37200 Tours, France

Received 1 October 2000; accepted 1 May 2001

Abstract

Consider the 2-machine flowshop scheduling problem with the objective of minimizing both the total completion time and the makespan criteria. The latter is assumed to be optimized prior to the former. In view of the $\mathcal{NP}$-hardness of the problem an Ant Colony Optimization approach is proposed to solve it. The heuristic also uses feature of Simulated Annealing search and local search algorithms. Computational experiments show its effectiveness compared to existing heuristics. The extension to the total completion time problem is also studied.
© 2002 Published by Elsevier Science B.V.

Keywords: Scheduling; Multiple objective programming; Ant Colony Optimization

1. Introduction

Consider the 2-machine flowshop scheduling problem with n jobs to schedule. It is assumed that the processing of a job cannot be interrupted. Let the processing times of job i be referred to as a_i on the first machine and b_i on the second machine. We note C_i the completion time of job i on machine 2 where $i = 1, \ldots, n$. The makespan criterion, noted $C_{\max}$, is defined as the maximum completion time of jobs on machine 2 whilst the total completion time criterion, noted $\sum C_i$, is defined as the sum of completion time of jobs on machine 2. We assume that the total completion time criterion has to be minimized subject to the condition that the makespan computed is minimum. As these two criteria are regular performance measure, the search for an optimal schedule can be restricted to the set of permutation schedules. This problem is referred to as a lexicographical minimization and is noted $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ [25]. Whilst the $F2\|C_{\max}$ problem is polynomially solvable by the so-called Johnson's algorithm [15], the $F2\|\sum C_i$ problem is known to be strongly $\mathcal{NP}$-hard. The lexicographical problem considered is also strongly $\mathcal{NP}$-hard [1].

Multiple criteria scheduling problems have been subject to a growing interest in the last decade. Some earlier surveys [6,8,14,17] consider a basic decomposition of such problems and mainly focus on single machine problems. Multicriteria scheduling literature has recently been revisited according to multicriteria optimization concepts [25].

* Corresponding author. Tel.: +33-2-47-36-14-14; fax: +33-2-47-36-14-22.

E-mail address: tkindt@e3i.univ-tours.fr (V. T'kindt).

0377-2217/02/$ - see front matter © 2002 Published by Elsevier Science B.V.
PII: S0377-2217(02)00265-5

A new classification and some methods to solve multicriteria scheduling problems are presented. The $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ problem has been extensively studied in the literature and both exact [11,21,26] and heuristic algorithms [10–12,19,21, 26] have been proposed. The two most effective heuristics are called INSERT [11] and TGB [26]. Among the exact algorithms the most efficient one is a branch-and-bound algorithm proposed by T'kindt et al. [26] which can handle problems up to 30 jobs in the hardest cases. The two other existing exact algorithms are limited to 10 jobs in a reasonable time. The $F2\|F_l(C_{\max}, \sum C_i)$ problem, where $F_l(C_{\max}, \sum C_i)$ stands for the convex combination of both criteria, has also been studied [18,23,27] and heuristic as well as exact algorithms have been proposed. The enumeration of the efficient schedules has been tackled by Sayin and Karabati [22] and the exact algorithm presented is able to solve problems up to 20 jobs in the hardest cases.

Purpose of this work is to present an Ant Colony Optimization (ACO) algorithm to solve the lexicographical problem. Since the last decade, ACO algorithms become more and more used in OR field [7] like for solving the traveling salesman problem [2] or the quadratic assignment problem [9]. However, few applications in scheduling literature have been encountered [3,24]. The basic idea of ACO algorithms come from the ability of ants to find shortest paths from their nest to food locations. Considering a combinatorial optimization problem, an ant iteratively builds a solution of the problem. This constructive procedure is conducted using at each step a probability distribution which corresponds to the pheromone trails in real ants. Once a solution is completed, pheromone trails are updated according to the quality of the best solution built. Hence, cooperation between ants is performed by the common structure which is the shared pheromone matrix. We propose such a population based heuristic to solve our lexicographical problem. Besides, intensification is emphasized using a local search procedure whilst diversification is emphasized using feature from Simulated Annealing search.

The remainder of the paper is organized as follows. In Section 2 we outline the proposed ACO heuristic. Section 3 contains the computational experiments for both the $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ and $F2\|\sum C_i$ problems. Section 4 deals with a discussion on the stability of the heuristic.

2. The SACO heuristic

Before presenting the proposed ACO heuristic, we focus on a result linked to the *Adjacent Pairwise Interchange* (API) neighborhood operator.

Lemma 2.1 [13]. *Given an initial schedule, it is possible to compute for the* $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ *problem an optimal schedule by applying a fixed number of jobs API.*

Proof. As a schedule for the $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ is a permutation schedule, i.e. a list of consecutive jobs, the graph where all nodes are possible schedules and vertices are API operations, is strongly connected. We can deduce that an optimal schedule is reachable from any feasible schedule and some API operations. □

Instead of the API operator, the k-API operator is considered in this paper. It is defined as follows: consider a fixed value k, a schedule S and a position $i < n - k$ in this schedule. From applying the k-API operation on S at position i we obtain a schedule S' where the jobs in position i and $i + k$ in S have been exchanged. Obviously Lemma 2.1 still holds for the k-API operator. It emphasizes the idea of using a multiple start point metaheuristic, such as an ACO heuristic, combined with a local search using this operator.

The ACO heuristic proposed to solve the $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ problem is referred to as SACO. In that heuristic each ant builds a feasible schedule by using a constructive procedure. This procedure uses a memory shared by the colony and which captures the ways to build a good feasible schedule. This memory is called the *pheromone matrix*. Let τ be this matrix and $\tau_{i,j}$ the probability of having job i at position j in a good schedule for the $\sum C_i$ criterion. $\tau_{i,j}$ is referred to as the *pheromone trail*. Starting from position 1 to position n, the most suitable job for position j is chosen

according to either the *intensification* mode or the *diversification* mode. We note p_0 the selection probability of being in one of these two modes. Let σ be the $j-1$ first scheduled jobs. In the intensification mode, an ant chooses as the most suitable job for position j, the one with the highest value of $\tau_{i,j}$ such that it exists at least one optimal schedule for the makespan beginning with σi. This *makespan check* can be done using Johnson's algorithm for the $F2\|C_{\max}$ problem. Let S be the schedule obtained by applying that algorithm on the set of jobs not in σi. If the value of the makespan for the schedule σiS is equal to the optimal value, then it exists at least one optimal schedule for the makespan that begins with subsequence σi. In the diversification mode, an ant uses a wheel process to select the most suitable job. This procedure is the same than in classical genetic algorithms except that only a job satisfying the makespan check described before, can be chosen. When an ant has built a complete schedule, a local search is applied. This one is performed as follows. For each position j of the schedule, compute $n-j$ schedules by applying the $1\text{-API}, 2\text{-API}, \ldots, (n-j)\text{-API}$ operators. Among these schedules plus the starting one, keep the schedule that has an optimal value of the makespan and the lowest value for the total completion time criterion. Therefore, consider position $j+1$. This local search has an overall $O(n^3)$ complexity. After all the schedules have been built by the ants, the best one is kept and the pheromone matrix is updated using the evaporation and enforcement processes. The former decreases the pheromones trails by setting them to $\rho\%$ of their previous value whilst the latter increases the pheromone trails that correspond to the scheduled kept at the current iteration. The heuristic terminates if the number of iterations exceeds the total number of iterations allowed.

Using a local search algorithm within the ACO heuristic emphasizes the convergence of the algorithm since few different schedules may be considered at the end of each iteration. To regulate the use of diversification and intensification processes we modify the scheme of classical ACO heuristics and consider that diversification is preferred at the beginning of the resolution whilst intensification is preferred at the end. That principle is equivalent to the acceptance probability controlling in Simulated Annealing search. In the heuristic proposed, it means that the selection probability p_0 is no longer fixed along the resolution. Let N be the total number of iterations before stopping. At an iteration k, the selection probability is defined by $p_0 = \log(k)/\log(N)$. Hence for an ant, lower is the value p_0 and higher is the chance to choose a job at position j using the diversification mode. A straight consequence is that it is no longer necessary to initially randomly generate the pheromone trails: as diversification is enforced at the beginning of the resolution, it is sufficient to consider equal initial values for the $\tau_{i,j}$'s. As in the ACO heuristic proposed by Stützle [24], the values $\tau_{i,j}$ belong to an interval $[\tau_{\min}; \tau_{\max}]$ to avoid pheromone trails from being negligible. The heuristic proposed is summarized in Fig. 1. The values of the bounds $\tau_{\min}$ and $\tau_{\max}$ and evaporation coefficient ρ are set as in the ACO heuristic proposed by Stützle for the $F\|C_{\max}$ problem.

The following lemma establishes the computational complexity of the proposed procedure.

Lemma 2.2. *The SACO heuristic for the* $F2\|\mathrm{Lex}(C_{\max}, \sum C_i)$ *problem requires* $O(n^3)$ *time.*

Proof. Recall that the local search used also requires $O(n^3)$ time. At a given position, the choice of the job to schedule can be done in $O(n^2)$ time as we have $O(n)$ possible jobs and the makespan check can be implemented in $O(n)$ time. Hence, the constructive procedure to get a schedule runs in $O(n^3)$ time and the whole heuristic requires $O(NMn^3)$ time. As N and M are data, the complexity of the heuristic is $O(n^3)$. □

Lemma 2.2 states that the complexity of the SACO heuristic is equivalent to that of the local search used. Indeed, the practical average complexity of the latter is lower than the one of the former.

3. Computational experiments

In this section we present the results of a series of computational experiments conducted to test

The SACO heuristic

/* Let T be the set of jobs to schedule */

/* Let N be the number of iterations and M the number of ants */

Step 1: $\rho = 0.9$; $\tau_{max} = \frac{1}{1-\rho}$;

$\tau_{min} = \tau_{max}/5$; $\tau_{i,j} = \tau_{max}$, $\forall i, j = 1..n$; $S_{best} = (1; 2; \ldots; n)$;

Step 2: For $Iteration$=1 to N Do

$p_0 = \frac{\log(Iteration)}{\log(N)}$;

For Ant=1 to M Do

$L = T$; $S_{ant} = \emptyset$;

For $Position$=1 to n Do

Generate a random number $0 \leq p \leq 1$;

If $(p \geq p_0)$ Then

/* Diversification mode */

Using the values $\tau_{i,Position}$, $i \in L$, apply a wheel and the *makespan check* to select the job k to schedule at current $Position$;

Else

/* Intensification mode */

Choose the job k with the highest value $\tau_{i,Position}$, $i \in L$, that satisfies the *makespan check*, to schedule at current $Position$;

End If

$S_{ant}[Position] = k$; $L = L - \{k\}$;

End For

Start the local search on S_{ant} to improve it;

End For

Let S be the best schedule for the $\sum C_i$ criterion computed by the ants;

$\tau_{i,j} = \rho\tau_{i,j}$, $\forall i, j = 1..n$; $\tau_{S[j],j} = \tau_{S[j],j} + \frac{1}{\sum C_i(S)}$, $\forall j = 1..n$;

If S improves S_{best} for the $\sum C_i$ criterion, set $S_{best} = S$;

End For

Step 3: Display S_{best}, $C_{max}(S_{best})$ and $\sum C_i(S_{best})$;

Fig. 1. An ant heuristic to solve the $F2||\mathrm{Lex}(C_{\max}, \sum C_i)$ problem.

the effectiveness of the SACO heuristic. The experiments were designed to determine the average and worst-case performance of the heuristics INSERT, TGB and SACO on large problems. Since it is impractical to solve large problems optimally, the heuristics are compared to the best known upper bound, i.e. the best result given by these three heuristics. Nevertheless, for small problems comparisons with a branch-and-bound algorithm [26] are conducted to evaluate the deviation of the heuristics from the optimal solution. All the algorithms were coded in C and run on a PC

Table 1
Comparisons to the best known upper bound for large problem size

n	INSERT				TGB				SACO			
	δ_{avg}	δ_{max}	t_{avg}	#	δ_{avg}	δ_{max}	t_{avg}	#	δ_{avg}	δ_{max}	t_{avg}	#
50	0.73	2.18	0.27	1	0.55	1.90	0.06	4	0.01	0.27	4.02	45
80	0.64	1.40	2.44	1	0.26	1.38	1.03	15	0.04	0.25	16.62	34
110	0.52	1.85	10.54	1	0.24	2.03	3.63	15	0.03	0.35	39.62	34
140	0.40	1.13	32.43	4	0.13	0.53	8.59	20	0.05	0.31	78.05	26
170	0.46	1.01	82.14	2	0.23	1.17	21.73	19	0.06	0.31	138.23	29
200	0.39	1.03	179.67	1	0.25	1.25	45.31	18	0.04	0.27	223.64	31

Pentium 400 MHz under the Windows operating environment. Concerning the tuning of the values N (number of iterations) and M (size of the colony), some earlier experiments [16] show that the most suitable tradeoff between efficiency and effectiveness is obtained with $N = 100$ and $M = 20$.

Processing times were randomly generated using a discrete uniform distribution between 1 and 100. Large problems consist of 50, 80, 110, 140, 170, and 200 jobs whilst for small problems we consider 10, 15, 20, and 25 jobs. Fifty instances for each problem size n were solved. Table 1 provides the average (δ_{avg}) and maximum (δ_{max}) deviations of the heuristic solution from the best known upper bound (best result among INSERT, TGB and SACO solutions) as a percentage of this best known upper bound. These deviations are computed according to the total completion time criterion since all the heuristics compute a schedule that has an optimal value of the makespan. Table 1 also provides the average computational time (t_{avg} in seconds) and the number of instances where a heuristic is the only one to give the best solution (#).

The following observations can be made from Table 1.

1. Concerning the $\sum C_i$ criterion, SACO outperforms both INSERT and TGB.
2. Concerning the computational time, TGB outperforms both INSERT and SACO. However, for higher values of n, SACO may be quicker than INSERT since the increase in the computational time is higher for INSERT than for SACO.
3. The average computational time of TGB seems to grow quicker than that of SACO.

Observation 1 is justified by both the average and maximum deviations and the number of instances where it gives the best result. Observations 2 and 3 stand that for problem size lower than 200, SACO is the slowest heuristic. Nevertheless, it appears that for higher values SACO may be the quickest. It is consistent with the complexity of the INSERT heuristic, which requires $O(n^4)$ time, whilst SACO requires $O(n^3)$. The complexity of the TGB heuristic can be hardly stated. It is made up of two stages: the first one is a greedy procedure that requires $O(n^3)$ time whilst the second one is a local search.

We also examined the impact of the SACO heuristic on the exact resolution of small size problems. We referred to the heuristic given by INSERT and TGB heuristics as $U1$ (i.e. $U1$ heuristic returns the best schedule, computed by either INSERT or TGB heuristics). $U2$ referred to the INSERT, TGB and SACO heuristics. Table 2 provides the average (δ_{avg}) and maximum (δ_{max}) deviations of the $U1$ and $U2$ solutions from the optimal value of the total completion time as a percentage of that optimal value. Table 2 also provides the average computational time (t_{avg} in seconds) of the branch-and-bound algorithm, as well as the average number (d_{avg}) of nodes evaluated when the solution of the corresponding heuristic (either $U1$ or $U2$) is used as the initial upper bound.

Concerning the $\sum C_i$ criterion, SACO reduces significantly the average and maximum deviations of the best known upper bound from the optimal solution. However the computational time saved in the branch-and-bound algorithm is low. The average number of nodes saved is about 2.8%.

Table 2
Comparisons to the optimal completion time value for small problem size

n	$U1$				$U2$			
	δ_{avg}	δ_{max}	t_{avg}	d_{avg}	δ_{avg}	δ_{max}	t_{avg}	d_{avg}
10	0.32	3.19	0.15	335	0.00	0.00	0.14	329
15	0.54	4.71	2.11	4599	0.01	0.21	1.94	4201
20	0.60	4.14	40.87	51,490	0.07	0.55	39.03	45,608
25	0.39	1.67	932.21	487,297	0.08	0.39	892.20	473,549

The SACO heuristic can be easily modified to solve the $F2||\sum C_i$ problem as only the *makespan check*, when choosing a job, has to be removed. The $F2||\sum C_i$ problem has been studied in the literature and the best heuristic, noted BSR, and exact procedures have been proposed in [4]. In Table 3 we report the same computational experiments than in Table 1.

From Table 3 we observe that the SACO heuristic outperforms the BSR heuristic. The strength of the BSR heuristic is its low average computational time. Despite the fact that the BSR heuristic also requires $O(n^3)$ time [4,5], its practical average complexity is significantly lower than the one of SACO. Concerning comparisons with the branch-and-bound algorithm proposed in [4], we have conducted the same computational experiments than in Table 2. For 30 jobs problem size, the average number of nodes saved was about 24.3% when considering both BSR and SACO heuristics as an upper bound for the exact algorithm (instead of only BSR). The time spent by this one was reduced from 3 minutes and was about 15 minutes. Besides, the average deviation of the best upper bound, given by either BSR or SACO, was lower than 0.1% (instead of 0.5% for BSR). The maximum deviation has been reduced from 1.85% to 0.44%.

4. Discussion

One of the major disadvantage of the SACO heuristic is linked to its stability. Like heuristics using random numbers, two runs of the SACO heuristic on the same instance may not give the same results. To strengthen the results provided in Section 3 it would be interesting to give an evaluation of their reliability. One way to proceed is to consider a statistical evaluation of the stability of the SACO heuristic. Fisher's coefficient is a measure of the degree of flattening of a frequency curve near its mode, i.e. its most frequent value [20]. To evaluate the stability of the SACO heuristic we first study the frequency curve of the total completion time values computed. Let α_F be Fisher's coefficient defined as follows: $\alpha_F = \mu_4/\sigma^4$ where μ_4 is the fourth moment and σ^4 the standard deviation power 4. Consider a problem instance of the lexicographical problem and let $\sum C_i^p$ be the

Table 3
Comparisons to the best known upper bound for large problem size

n	BSR				SACO			
	δ_{avg}	δ_{max}	t_{avg}	#	δ_{avg}	δ_{max}	t_{avg}	#
50	0.26	0.99	0.12	6	0.00	0.05	3.51	43
80	0.24	0.85	0.39	4	0.01	0.31	13.30	46
110	0.16	0.53	0.86	9	0.02	0.36	32.60	41
140	0.16	0.55	1.59	12	0.03	0.24	66.80	38
170	0.15	0.66	2.71	7	0.01	0.23	120.05	43
200	0.15	0.57	4.03	10	0.01	0.13	183.42	40

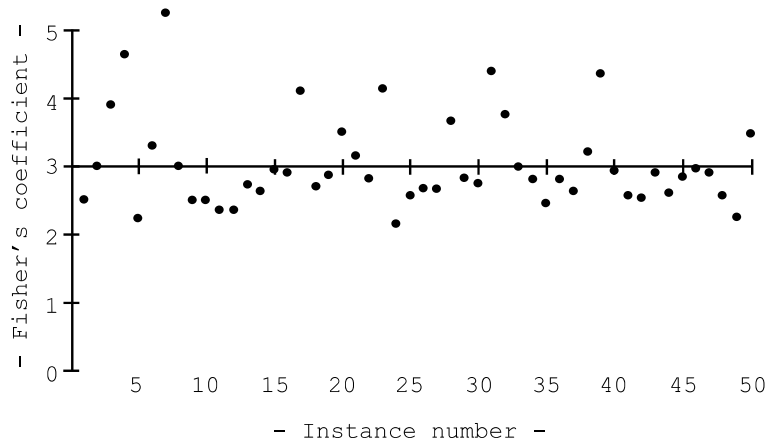

Fig. 2. Values of Fisher's coefficient for problems with 100 jobs.

value of the total completion time computed by the SACO heuristic when ran for the pth time on that instance. Let $\sum C_i^{\text{avg}}$ be the average value over the P values $\sum C_i^p$. The standard deviation is given by

$$\sigma = \sqrt{\frac{1}{P}\sum_{p=1}^{P}\left(\sum C_i^p - \sum C_i^{\text{avg}}\right)^2}$$

and the fourth moment by

$$\mu_4 = \frac{1}{P}\sum_{p=1}^{P}\left(\sum C_i^p - \sum C_i^{\text{avg}}\right)^4.$$

If for all the runs of the SACO heuristic on that instance, all the values $\sum C_i^p$ tends to be equal, then α_F tends to ∞. Unfortunately, it may not often appear. In that case, the only information we can obtain is the following: if $\alpha_F = 3$ then we can assume without loss of generality that the distribution law of the $\sum C_i^p$ values follows a normal distribution. If α_F tends to 0 the frequency curve is assumed to be flat, whilst if $\alpha_F > 3$ this curve can be assumed non flat around the mode.

Some computational experiments have been conducted to evaluate Fisher's coefficient. We considered problems consisting of $10, 20, 30, \ldots, 100$ jobs. For each problem size, 50 instances were generated and for each instance the SACO heuristic has been ran 50 times. It would not be suitable to present all the obtained results so we restrict to the worst results, given for problems with 100 jobs. The 50 Fisher's coefficient values are reported in Fig. 2.

Clearly, the values are near 3 and hence, the distribution of the total completion time values given by the SACO heuristic can be considered as a normal distribution (which is not a flat distribution). Moreover the standard deviation for that problem size is always lower than 0.22% of the average total completion time value. Regarding to that deviation, we can conclude that about 95% of the total completion time values are around 0.22% of the average value. Hence, the stability of the SACO heuristic can be stated.

5. Conclusion

In this paper an Ant Colony Optimization heuristic for the $F2\|\text{Lex}(C_{\max}, \sum C_i)$ problem, SACO, has been presented. The overall complexity of this heuristic is $O(n^3)$. Extensive computational experiments suggest that SACO yields better results than existing heuristics. Despite the fact that for problem size lower than 200 the TGB heuristic is the quickest one, it is guessed that for larger problems SACO becomes the most efficient. The extension of the SACO heuristic to the $F2\|\sum C_i$ problem still remains more effective than existing heuristics for that problem.

The reliability of the results is emphasized by a statistical study of the distribution of the total completion time values computed by the SACO heuristic along several runs on the same instance. The conclusion is that this heuristic can be consider as a stable one. The local search used in the heuristic may influence that conclusion.

References

[1] C.-L. Chen, R.L. Bulfin, Complexity of multiple machines, multicriteria scheduling problems, in: Third Industrial Engineering Research Conference (IERC'94), Atlanta (USA), 1994, pp. 662 –665.

[2] A. Colorni, M. Dorigo, V. Maniezzo, Distributed optimization by ants colonies, in: F. Varela, P. Bourgine (Eds.), First European Conference on Artificial Life, 1991, pp. 134–142.

[3] A. Colorni, M. Dorigo, V. Maniezzo, M. Trubian, Ant system for job-shop scheduling, Belgian Journal of Operations Research Statistics and Computer Science (JORBEL) 34 (1994) 39–53.

[4] F. DellaCroce, M. Ghirardi, R. Tadei, An improved branch and bound algorithm for the two-machine total completion time flow shop problem, European Journal of Operational Research 139 (2) (2002) 293–301.

[5] F. Della Croce, M. Ghirardi, R. Tadei, Recovering Beam Search: A hybrid heuristic method for combinatorial optimization problems. Submitted to Operations Research, 2002.

[6] P. Dileepan, T. Sen, Bicriterion static scheduling research for a single machine, Omega 16 (1) (1988) 53–59.

[7] M. Dorigo, G. Di Caro, L.M. Gambardella, Ant algorithms for discrete optimization, Artificial Life 5 (3) (1999) 137–172.

[8] T.D. Fry, R.D. Armstrong, H. Lewis, A framework for single machine multiple objective sequencing research, Omega 17 (6) (1989) 595–607.

[9] L.M. Gambardella, E. Taillard, M. Dorigo, Ants colonies for the qap, Journal of the Operational Research Society 5 (1999) 167–176.

[10] J.N.D. Gupta, K. Hennig, F. Werner, Local search heuristic for the two-stage flowshop problems with secondary criterion, Computers and Operations Research 29 (2) (2002) 113–149.

[11] J.N.D. Gupta, V.R. Neppalli, F. Werner, Minimizing total flow time in a two-machine flowshop problem with minimum makespan, International Journal of Production Economics 69 (3) (2001) 323–338.

[12] J.N.D. Gupta, N. Palanimuthu, C.-L. Chen, Designing a tabu search algorithm for the two-stage flowshop problem with secondary criterion, Production Planning and Control 10 (3) (1999) 251–265.

[13] J.N.D. Gupta, F. Werner, On the solution of 2-machine flow and open shop scheduling problems with secondary criteria, in: 15th ISPE/IEE International Conference on CAD/CAM, Robotics, and Factories of the Future, Aguas de Lindoia (Brasil), 1999, pp. MW4.1-MW4.6.

[14] J.A. Hoogeveen, Single-Machine Bicriteria Scheduling, Ph.D. Thesis, CWI, Amsterdam, 1992.

[15] S.M. Johnson, Optimal two and three stage production schedules with set-up time included, Naval Research Logistics Quarterly 1 (1954) 61–68.

[16] D. Laügt, F. Tercinet, N. Monmarché, V. T'kindt, Une heuristique basée sur les colonies de fourmis pour résoudre un problème d'ordonnancement bicritère, Research report 231, E3i/Université de Tours (France), in French, 2000.

[17] A. Nagar, J. Haddock, S.S. Heragu, Multiple and bicriteria scheduling: A literature survey, European Journal of Operational Research 81 (1995) 88–104.

[18] A. Nagar, S.S. Heragu, J. Haddock, A branch-and-bound approach for a two-machine flowshop scheduling problem, Journal of the Operational Research Society 46 (1995) 721–734.

[19] V.R. Neppalli, C.-L. Chen, J.N.D. Gupta, Genetic algorithms for the two-stage bicriteria flowshop problem, European Journal of Operational Research 95 (1996) 356–373.

[20] N.A. Rahman, A course in theoretical statistics, Griffin (London), 1968.

[21] C. Rajendran, Two-stage flowshop scheduling problem with bicriteria, Journal of the Operational Research Society 43 (9) (1992) 871–884.

[22] S. Sayin, S. Karabati, A bicriteria approach to the two-machine flow shop scheduling problem, European Journal of Operational Research 113 (1999) 435–449.

[23] F.S. Serifoglu, G. Ulusoy, A bicriteria two-machine permutation flowshop problem, European Journal of Operational Research 107 (1998) 414–430.

[24] T. Stützle, An ant approach to the flow shop problem, in: Proceedings of EUFIT'98, Aachen (Germany), 1998, pp. 1560–1564.

[25] V. T'kindt, J.-C. Billaut, Multicriteria Scheduling: Theory, Models and Algorithms, Springer, forthcoming (2002).

[26] V. T'kindt, J.N.D. Gupta, J.-C. Billaut, Two-machine flowshop scheduling with a secondary criterion, Computers and Operations Research, forthcoming (2002).

[27] W.-C. Yeh, A new branch-and-bound approach for the $n/2/\text{flowshop}/\alpha f + \beta c_{\max}$ flowshop scheduling problem, Computers and Operations Research 26 (1999) 1293–1310.