Int J Adv Manuf Technol (2007) 33: 793–802
DOI 10.1007/s00170-006-0492-8

ORIGINAL ARTICLE

Kuo-Ching Ying · Shih-Wei Lin

Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems

Received: 30 June 2005 / Accepted: 28 October 2005 / Published online: 11 April 2006

Abstract Ant colony optimization (ACO) is a novel intelligent meta-heuristic originating from the foraging behavior of ants. An efficient heuristic of ACO is the ant colony system (ACS). This study presents a multi-heuristic desirability ACS heuristic for the non-permutation flowshop scheduling problem, and verifies the effectiveness of the proposed heuristic by performing computational experiments on a well-known non-permutation flowshop benchmark problem set. Over three-quarters of the solutions to these experiments are superior to the current best solutions in relevant literature. Since the proposed heuristic is comprehensible and effective, this study successfully explores the excellent potential of ACO for solving non-permutation flowshop scheduling problems.

Keywords Scheduling · Non-permutation flowshop · Ant colony optimization

1 Introduction

Scheduling is an effective means of allocating resources optimally. Efficient scheduling is critical to numerous industries, having become essential for survival in the intensely competitive business environment. A well-known NP-hard problem is flowshop scheduling problems (FSP). Although extensively studied for over four decades, FSP has not been precisely defined [1]. However, the following general characterizations of FSP are common in pertinent literature. Each job comprises *m* operations; each operation requires a different machine; all jobs are processed in the same processing sequence; and the job sequence of each machine has to be identified to minimize (or maximize) a given objective function. The job sequence of each machine remains unchanged in a permutation FSP, while a problem where the sequence changes is called a non-permutation FSP.

The flowshop system has been adopted in the electronics manufacturing and chemical processing sectors. Relevant literature contains numerous theoretical studies on FSP, most of which have the objective of minimizing the maximum completion time (makespan) [2]. This focus occurs because the makespan is the easiest criterion to adopt, and also indicates other benefits, such as maximizing the machine utilization of a production line [1]. Consequently, minimizing makespan is the main objective of this study.

Reisman et al. [3] reviewed pertinent literature of FSP and performed a statistical correlation analysis. Several investigations have also experimentally compared various algorithms [1, 4–6]. FSP is an NP-complete combinatorial optimization problem involving more than two machines [7]. Hence, even a simple FSP may require considerable computation by exact methods. Therefore, in practice, practitioners often seek heuristic methods that generate near-optimum solutions at low computational cost.

The formidable computational requirements of FSP have resulted in numerous attempts to develop heuristic procedures, leading to interest in a new class of algorithms called metaheuristics. The metaheuristic algorithmic framework can be applied to various optimization problems with slight modifications. Examples of metaheuristic algorithms include genetic algorithms, simulated annealing, and taboo search. Relevant literature indicates that these methods have occupied a growing space in FSP and can obtain good results [8–10]. Nevertheless, almost all published heuristics focus on permutation schedules [2], schedules that can be specified simply by providing a permutation of jobs are simple to combine. Potts et al. [11] have indicated that this simplicity is achieved at the price of significantly inferior schedules.

K.-C. Ying (✉)
Department of Industrial Engineering and Management Information, Huafan University,
Taipei, Taiwan, Republic of China
e-mail: kcying@huafan.hfu.edu.tw
Tel.: +886-2-26632102
Fax: +886-2-26633981

S.-W. Lin
Department of Information Management, Huafan University,
Taipei, Taiwan, Republic of China

This study examines the potential of ant colony optimization (ACO) for the non-permutation FSP. ACO is a metaheuristic algorithm for solving discrete optimization problems [12]. The first version of the ACO metaheuristic, known as the ant system (AS), was presented by Colorni et al. [13], and was further developed in a PhD thesis by Dorigo [14]. AS is inspired by ant colony foraging behavior. A colony of ants can identify the shortest pathway from a food source to their nest without using visual cues; they communicate by pheromone, an aromatic substance. While searching, ants secrete pheromone on the ground and generally follow pheromone previously laid by other ants. Ants are most likely to follow pathways marked by large accumulations of pheromone from other ants that have previously followed that route. Since short pathways have higher traffic densities than long pathways, they accumulate pheromone more rapidly. Consequently, ants favor shorter pathways over longer ones.

Pertinent literature indicates that ACO can yield excellent results for various NP-hard combinatorial optimization problems, such as the traveling salesman [15], sequential ordering [16], quadratic assignment [17], vehicle routing [18], graph coloring [19], partitioning [20], power system optimization [21] and telecommunications network [22].

Ant colony system (ACS) is an efficient ACO heuristic. This study presents a multi-heuristic desirability ant colony system (MHD-ACS) heuristic for the *n*-job, *m*-machine non-permutation FSP to minimize the makespan. Some previous publications have focused on applying ACO to different scheduling environments, such as the single machine [23, 24], permutation flowshop [25, 26], jobshop [27] and openshop [28]. Nevertheless, this study is apparently the first to adopt ACS for non-permutation FSPs. The *n*-job, *m*-machine non-permutation FSP with minimizing makespan (C_{max}) can be presented using the classical notation as $F||C_{\max}$ [29]. The $F||C_{\max}$ problem generally makes the following assumptions:

- The number of jobs, and their processing times on each machine, are known in advance.
- The number of machines is known, and all machines are continuously available.
- The setup times of the operations are included in the processing times, and are sequencing-independent.
- All the jobs are available at start time.
- Every job is processed on a maximum of one machine at a time without preemption.
- Every machine can process no more than one job at a time.
- Every job must be processed once on each machine and in the same machine order, for example, $M_1, M_2, \cdots, M_m$.

2 Development of MHD-ACS heuristic

The problem of implementing a multi-heuristic desirability ant colony system (MHD-ACS) for $F||C_{\max}$ is transformed into the following disjunctive graph model in advance.

Representing $F||C_{\max}$ by a disjunctive graph model

The $F||C_{\max}$ problem can be reduced to a graph model. Individual instances of this problem can be associated with a disjunctive graph *G=(O,C,D)*, where *O* denotes a set of nodes; *C* represents a set of conjunctive directed arcs, and *D* is a set of disjunctive undirected arcs (see Fig. 1). The nodes of *O* represent all of the processing operations O_{ij} (the *j*th operation of job *i*; $i = 1, 2, \cdots, n; j = 1, 2, \cdots, m$) that act upon the *n* jobs; *C* denotes the precedence relationships between the processing operations of each job, and *D* represents the machine constraints of operations belonging to different jobs. Moreover, two dummy nodes, the nest *N* and food source *F* exist, where *N* has conjunctive directed arcs emanating to the first operations of the *n* jobs, and *F* has conjunctive directed arcs originating from all the final operations of the *n* jobs. Therefore, a total of $(n \cdot m + 2)$ nodes are available, where all nodes of the same machine are linked in a pairwise manner in both directions. For conciseness, all the arrows of nodes in the figure that are linked in a pairwise manner in both directions are omitted.

Figure 1 shows an instance of the $F||C_{\max}$ problem involving three jobs and four machines. The disjunctive undirected (broken) arcs form four cliques, one for each machine, representing the sequence in which operations are performed on the machine. This instance clearly reveals that solving the $F||C_{\max}$ problem identifies a path of the disjunctive graph *G* in which all nodes have to be visited, thus minimizing the makespan.

The example in Fig. 2 helps visualize the possible successive decisions of an ant, as well as the final feasible solution to the $F||C_{\max}$ problem. An ant was positioned on the nest node *N* (see Fig. 2a) at the beginning of the process of obtaining a solution. In the figure, *U* denotes the set of nodes not yet visited, and *S* represents the set of nodes whose predecessors have previously been visited. Nodes belonging to *S* are labeled in the figure. Initially, $U = O + \{F\}$ and *S*

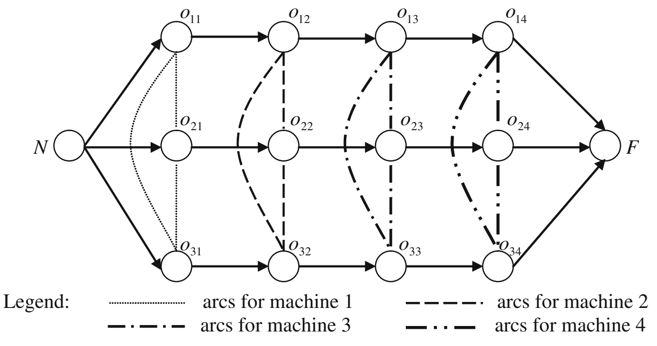
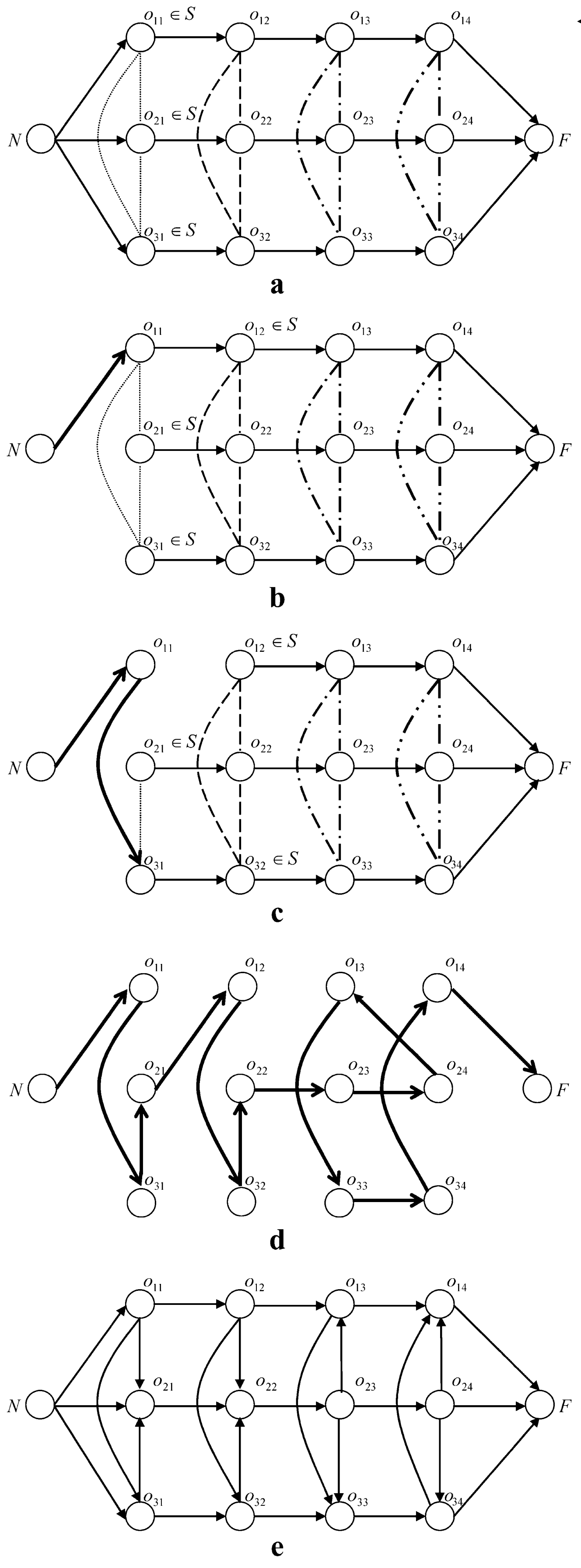


Fig. 1 An instance of the $F||C_{\max}$ problem



◀ **Fig. 2a–e** An example of an ant constructing a feasible solution

include all the nodes corresponding to the first operation of each job.

An ant selects the next moving node by the state transition rule when constructing the set of feasible routes. The selected node is then added to a taboo list and removed from *U* and *S* The passed arc is thickened, and useless arcs are removed. If the selected node is not the last in its job, then its immediate successor is appended to *S*. The process is iterated until *F* has been chosen. If the ant chooses to go to node O_{11}, thus bringing O_{12} into *S* then the result is as shown in Fig. 2b. If the ant proceeds to node O_{31} during the next step, then the result is as depicted in Fig. 2c. Finally, the assumed complete route is as illustrated in Fig. 2d. The node permutation $(O_{11}O_{31}O_{21}O_{12}O_{32}O_{22}O_{23}O_{24}O_{13}O_{33}O_{34}O_{14}F)$ of the above route can be derived from the taboo list. This route determines the sequence in which the operations are performed on each machine, and corresponds to the following acyclic-oriented graph (see Fig. 2e). The length of the longest path from *N* to *F* (i.e., the makespan of the corresponding feasible solution) can then be calculated.

Structure of the MHD-ACS heuristic

The proposed MHD-ACS heuristic links the length of an edge to the ranking index of the operation of a randomly selected constructive heuristic. The pheromone trail level is an adaptive memory, which represents the weight of the operation ranking index of the selected constructive heuristic, and is updated by the ants at runtime. Based on the analogy between ant colony foraging behavior and the process of solving the scheduling problem, the structure of MHD-ACS heuristic is as shown in Fig. 3. Three procedures are repeated until predetermined end conditions (e.g., maximum number of iterations, or maximum computation time allowed) are verified. First, the parameters $m, q_0, \alpha, \beta, \rho$ are set during the initialization step. For each edge (x,u), the initial pheromone trail level $\tau(x,u) = (n \cdot m \cdot UB)^{-1}$ is deposited, where *UB* denotes the upper bound of the optimum solution obtained from the benchmark problem set.

Second, a colony of *m* ants is initially positioned on the nest node during the main loop step. Each ant then randomly selects a constructive heuristic *z* according to the following probability distribution:

$$p_z = \frac{z_{best}}{N}$$

where Z_{best} is the number of the best solutions of the construction heuristic *z* for all test instances, and *N* is the number of the test instances.

Meanwhile, the heuristic desirability $\eta_z(x,u)$ for each edge (x,u) is set to the ranking index of operation *u* of the randomly selected constructive heuristic *z* Notably, $\eta(x,u)$ denotes the reciprocal of a cost measure (e.g. distance) from

```
/* Step 1: Initialization */
    Set the parameters $m$, $q_0$, $\alpha$, $\beta$, and $\rho$.
    For each edge $(x, u)$, the initial pheromone trail level
    $\tau(x,u) = (n \cdot m \cdot \mathrm{UB})^{-1}$ is deposited.

/* Step 2: Main loop */
    a  A colony of $m$ ants is initially positioned on nest node.
    b  Build a schedule for each ant as follows:
       Loop
           The heuristic desirability $\eta_z(x, u)$ for each edge $(x, u)$
           is set to the ranking index of operation $u$ of the
           randomly selected constructive heuristic $z$.
           Loop
               Each ant applies the state transition rule to select the
               next processing operation.
               Store the selected operation into taboo list.
           Until a schedule is constructed.
           Ants reduce the amount of pheromone on the edges they
           have visited by adopting the local updating rule.
       Until all ants have built complete schedules.
    c  The makespans of all constructed schedule are calculated.
    d  The global updating rule is applied to increase pheromone
       levels on the edges of the best schedule up to the current
       iteration, and to reduce pheromone levels on other edges.

/* Step 3: Stopping criterion */
    If either maximum number of iterations or the maximum allowed
    computation time is reached, then STOP; otherwise all the taboo
    lists are emptied and go to Step 2.
```

Fig. 3 The structure of MHD-ACS heuristic

nodes x to u in the original ACS, but is revised to $\eta_z(x,u)$ in the proposed approach, where it denotes the ranking index of operation u of the randomly selected constructive heuristic z.

Each ant then repeatedly applies the state transition rule to select the next processing operation until a complete schedule is constructed (i.e., all the nodes are visited). Both the heuristic desirability and pheromone quantity guide the schedule construction operations. The probability of being selected increases with both the heuristic desirability and the pheromone quantity. The selected operations are sequentially stored in a taboo list. During schedule construction, ants also reduce the amount of pheromone on the edges they have visited by adopting the local updating rule to shuffle the tour of other ants and avoiding local optima. The makespans of all constructed schedules are calculated once all ants have established complete schedules. The global updating rule is then adopted to increase pheromone levels on the edges of the best schedule up to the current iteration, and to reduce pheromone levels on other edges. This pheromone-modifying rule generally provides the most pheromone to edges selected by ants, causing all ants to converge to the best schedule.

Third, the procedure is iterated until it reaches either the maximum number of iterations or the maximum allowed computation time. If neither of these stopping criteria is reached, then all the taboo lists are emptied and the main loop step is iterated continuously. The following subsection discusses the relevant rules in detail.

State transition rule

While constructing a schedule, an ant k located at the current node of operation x selects the next operation y to which it should move by adopting the state transition rule given by

$$y = \begin{cases} \arg\max\limits_{u \in S_k(x)} \left\{ [\tau(x,\ u)] \cdot [\eta_z(x,\ u)]^{\beta} \right\} & \text{if } q \leq q_0 \\ Y & \text{otherwise} \end{cases} \tag{1}$$

where the desirability measure $\tau(x,u)$ is the cumulative pheromone trail of edge (x,u), which represents the weight of the ranking index of the randomly selected constructive heuristic, and is updated on execution by the ant cooperative learning mechanism. The heuristic desirability $\eta_z(x,u)$ denotes the ranking index of operation, u, of the randomly selected constructive heuristic z; $S_k\ (x)$ represents the set of feasible successive operations that remain to be visited by ant k positioned on node x; $\beta(\beta>0)$ is a parameter that determines the weight dimension of ranking index; q denotes a random number uniformly distributed in [0, 1], and $q_0 (0 \leq q_0 \leq 1)$ represents the probability of exploiting the best edge instead of other edges. Furthermore, Y is an operation that is randomly selected according to a probability distribution, termed the random-proportional rule, given by

$$p_k(x,\ y) = \begin{cases} \dfrac{[\tau(x,\ y)] \cdot [\eta_z(x,\ y)]^{\beta}}{\sum\limits_{u \in S_k(x)} [\tau(x,\ u)] \cdot [\eta_z(x,\ u)]^{\beta}} & \text{if } y \in S_k(x) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The state transition rule derived from Eqs. (1) and (2) tends to move to operations with edges that have large

Table 1 Computational results of simple constructive heuristics

Instance	UB	SPT	LWKR	LRM	SPT/TWKR	SPT TWK
flcmax_20_15_3	4437	6511	*4699*	4779	5515	5075
flcmax_20_15_6	4144	6083	*4585*	4681	5305	5368
flcmax_20_15_4	3779	5816	4579	*4143*	4988	5116
flcmax_20_15_10	4302	7340	*4437*	4631	5422	7458
flcmax_20_15_5	4373	6409	*4704*	4799	5347	5357
flcmax_20_20_1	4821	7846	*5010*	5036	5604	5796
flcmax_20_20_3	4779	6862	*5146*	5542	6249	5774
flcmax_20_20_9	4944	7665	5345	*5332*	6065	6172
flcmax_20_20_2	4886	8113	*5209*	5259	6145	6016
flcmax_20_20_10	4717	7172	5311	*5034*	6345	6092
flcmax_30_15_3	5226	8058	*5805*	5891	6976	6336
flcmax_30_15_4	5304	9549	6060	*5838*	7526	7073
flcmax_30_15_9	5079	8710	5699	*5509*	6665	6418
flcmax_30_15_8	5605	9384	6013	*5652*	6817	7014
flcmax_30_15_6	5147	8605	6097	*5806*	7327	7341
flcmax_30_20_3	6183	10539	*6426*	6507	7978	7976
flcmax_30_20_1	6037	9514	6980	*6851*	7432	7962
flcmax_30_20_6	6241	9163	6883	*6699*	7573	7781
flcmax_30_20_10	6095	9898	*6432*	6825	7792	7690
flcmax_30_20_2	5822	9755	*7042*	7048	8269	8268
flcmax_40_15_5	6986	10781	7701	*7024*	8412	8594
flcmax_40_15_9	6351	11601	*6978*	7000	8929	8481
flcmax_40_15_2	6506	10536	*7229*	7423	8888	8796
flcmax_40_15_10	6845	11431	7236	*7088*	8704	8049
flcmax_40_15_8	6783	11022	7533	*7437*	8801	8680
flcmax_40_20_3	7154	13565	*8006*	8247	10787	9339
flcmax_40_20_9	7528	12959	*7685*	8351	9765	8730
flcmax_40_20_6	7469	12980	*8342*	8538	9828	9559
flcmax_40_20_7	7608	13173	8195	*8133*	9561	9405
flcmax_40_20_5	7219	11699	*7773*	8471	9923	8841
flcmax_50_15_6	7673	12618	*8370*	8717	10742	9605
flcmax_50_15_5	7679	12759	*8208*	8595	10676	9766
flcmax_50_15_1	7416	13467	8724	*8132*	9823	10689
flcmax_50_15_8	7548	13727	*8285*	8818	10563	10079
flcmax_50_15_2	7750	13764	*8712*	8835	10540	9486
flcmax_50_20_2	8838	15676	*9809*	9858	11535	11295
flcmax_50_20_1	8539	14145	*9023*	9124	11122	10419
flcmax_50_20_7	8417	14802	9360	*9168*	11254	10930
flcmax_50_20_8	8590	12906	9290	*9109*	10177	11116
flcmax_50_20_4	8493	15191	*9436*	9615	11457	10324

ranking indices and ranking index weights in the randomly selected constructive heuristic. This selection process can generally be executed by a roulette wheel. Every time an ant k in operation x must select an operation y to which to move, it samples a random number q from uniform distribution [0, 1]. If $q \le q_0$, then the operation corresponding to the argument (arg) with maximum $[\tau(x,u)]\cdot[\eta_z(x,u)]^{\beta}$ is selected (exploitation), else an operation is selected according to Eq. (2) (exploration).

Local updating rule

An ant diminishes the pheromone level on its visited edges during schedule construction by the following local updating rule:

$$\tau(x,y) := (1-\rho)\cdot\tau(x,y) + \rho\cdot(n\cdot m\cdot UB)^{-1}$$

where ρ (0<ρ<1) denotes the local pheromone evaporating parameter.

The local updating rule dynamically changes the desirability of each edge, thus shuffling the tour. An ant is more likely to obtain an improved schedule if every ant explores a different schedule than if they all search in a narrow neighborhood of the current best schedule. Every time an ant constructs a schedule, the local updating rule is adopted to reduce the pheromone levels in the visited edges and reduce their attractiveness. Hence, the edges of one ant's schedule are less likely to be selected in constructing the schedules of other ants. Consequently, ants favor the exploration of edges not yet visited, preventing convergence to a common schedule.

Global updating rule

Global updating is performed after all ants have completed their schedules. To focus the search, pheromone levels at the edges of the current best schedule are increased, while pheromone levels at edges not belonging to the schedule are reduced. Pheromone levels are modified according to [15]:

$$\tau(x,y) := (1-\alpha)\cdot\tau(x,y) + \alpha\cdot\Delta\tau(x,y)$$

where

$$\Delta\tau(x,y) = \begin{cases} (L_{\text{best}})^{-1} & \text{if } (x,\ y) \in \text{incumbent best schedule} \\ 0 & \text{otherwise} \end{cases}$$

In the above equations, $\alpha (0<\alpha<1)$ denotes the pheromone evaporating parameter of global updating, and L_{best} represents the makespan of the current best schedule.

Selection of the heuristic desirability

The heuristic desirability $\eta(x,u)$ of ACS is defined as the reciprocal of a cost measure (e.g., distance) from node x to node u. In this study, the heuristic desirability is revised to $\eta_z(x,u)$, and is set as the ranking index of operation u of the randomly selected simple constructive heuristic (SCH) z.

The constructive heuristic is a single-pass method that constructs a schedule by fixing, at each step, the job position based on a simple rule in the sequence [11]. If the ranking indices of the randomly selected SCH z are ranked in non-ascending order, then the heuristic desirability from nodes (operations) $O_{ij}(i=1,2,\cdots,n; j=1,2,\cdots,m)$ and N to nodes (operations) $o_{kl}(k=1,2,\cdots,n;\ l=1,2,\cdots,m;\ k\neq i;\ l\neq j)$ are defined as the ranking indices of nodes O_{kl} (RI_{kl}), i.e., $\eta_z(o_{ij},o_{kl}) = RI_{kl}$; $\eta_z(N,o_{kl}) = RI_{kl}$. Otherwise, the reciprocal of the ranking indices of nodes o_{kl} (i.e., $\eta_z(o_{ij},o_{kl}) = 1/RI_{kl}$; $\eta_z(N,o_{kl}) = 1/RI_{kl}$) are adopted.

Several constructive heuristics applicable to the FSP have been assessed [30–32]. Since various parameters affect the relative performance of these heuristics, no single best SCH exists [33]. In the next section, 20 SCHs that have been evaluated in the literature are studied through further experiments. The Appendix shows the selected heuristics and their RI_{kl} formulae. James and Michael [30], Haupt [31] and Chang et al. [32] have all described these heuristics in detail.

3 Computational results and discussion

Test problems

The effectiveness of the proposed heuristic was verified by performing computational experiments on a well-known benchmark problem set established by Demirkol et al. [34] containing 600 randomly generated instances for six basic shop scheduling problems. This benchmark problem set allows researchers to compare their proposed algorithms with those of other researchers using an identical test problem set. The test problem set adopted herein was their 40 test instances for the $F\|C_{\max}$ problem, which were randomly generated as follows. All jobs are available at time zero, and the operation processing times are generated by a discrete uniform distribution between 1 and 200. The test instances involved two machine number values (m=15, 20) and four job number values (n=20, 30, 40, 50), resulting in eight combinations of m and n and a total number of operations ranging from 300 to 1000. The ratio of n to m varies between 1 and 3.3. These combinations yield a problem set which is not based on a specific application. Ten instances were generated for each of the eight combinations of m and n.

Since the size and complexity of the instances make exact solutions impractical, Demirkol et al. [34] solved each instance by five different constructive heuristics and three versions of the shifting bottleneck procedure. Extensive numerical research has indicated that the shifting bottleneck heuristic is extremely effective [35]. All algorithms were run on a SUN SPARC server 1000 Model 1104 with a 50 MHz processor, which is a multitasking system running under Unix. The upper bound (UB) for each instance was the best solution found by any of the algorithms. The computational time was

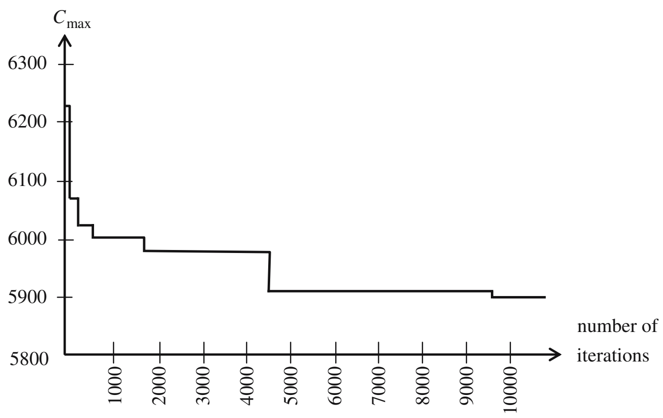


Fig. 4 Evolution of best solutions achieved in the flcmax-30-20-10 problem

recorded by the method that provided the best solution. If more than one algorithm found the best solution, then the algorithm with the shortest computational time was selected. The best available solutions for the test problem set in the literature appear to be those proposed by Demirkol et al. [34].

A lower bound (LB) can be obtained for each instance by relaxing the capacity constraints on all but one machine and solving to optimality the resulting single machine problem of minimizing makespan with release and delivery times [35]. Demirkol et al. [34] performed this method for each machine, and reported the highest makespan value obtained as a lower bound in each instance. To obtain a more compact and challenging set of test problems, Demirkol et al. ranked the instances according to decreasing order of percentage gap between the upper and lower bounds for each combination of m and n. Only the first five instances for each combination were finally presented. Thus, a total of 40 test instances were obtained for the $F\|C_{\max}$ problem as listed in Table 1, where the instances are denoted by the term *flcmax_n_m_Instance-Number*.

Table 2 Computational results of MHD-ACS and CHD-ACS

Instance	Benchmark problem			MHD-ACS		CHD-ACS	
	LB	UB	Time	$C_{\max}$	Time	$C_{\max}$	Time
flcmax_20_15_3	3354	4437	69.93	*4420*	46	4455	45
flcmax_20_15_6	3168	4144	0.09	*4044*	46	4124	45
flcmax_20_15_4	2997	*3779*	57.86	3786	46	3926	45
flcmax_20_15_10	3420	4302	66.68	*4265*	45	4298	44
flcmax_20_15_5	3494	4373	57.54	*4310*	45	4370	45
flcmax_20_20_1	3776	4821	159.12	*4819*	59	4878	57
flcmax_20_20_3	3758	4779	148.21	*4723*	60	5002	57
flcmax_20_20_9	3902	4944	175.74	*4922*	60	5023	56
flcmax_20_20_2	3881	4886	202.58	*4878*	60	4941	57
flcmax_20_20_10	3823	4717	164.07	*4715*	60	4932	56
flcmax_30_15_3	4020	5226	148.99	*5210*	93	5319	90
flcmax_30_15_4	4080	5304	163.22	*5284*	94	5473	89
flcmax_30_15_9	4022	5079	108.63	*5075*	95	5375	89
flcmax_30_15_8	4490	5605	0.17	*5593*	94	5603	90
flcmax_30_15_6	4184	*5147*	147.33	5149	93	5421	89
flcmax_30_20_3	4806	6183	0.24	*5987*	121	6059	116
flcmax_30_20_1	4772	6037	470.73	*5989*	124	6238	117
flcmax_30_20_6	5004	6241	394.33	*6195*	124	6512	116
flcmax_30_20_10	4899	6095	320.22	*5923*	121	6105	115
flcmax_30_20_2	4757	*5822*	388.40	5840	123	6255	117
flcmax_40_15_5	5560	6986	155.97	*6972*	154	7219	144
flcmax_40_15_9	5119	6351	223.64	*6310*	154	6417	145
flcmax_40_15_2	5290	*6506*	289.02	6532	154	6956	144
flcmax_40_15_10	5596	6845	186.49	*6712*	156	6913	145
flcmax_40_15_8	5576	6783	0.24	*6771*	156	6783	146
flcmax_40_20_3	5693	7154	615.49	*7132*	210	7564	198
flcmax_40_20_9	5998	7528	645.20	*7496*	208	7698	197
flcmax_40_20_6	5990	*7469*	673.92	7476	209	8103	198
flcmax_40_20_7	6170	7608	681.51	*7588*	207	7884	197
flcmax_40_20_5	6011	7219	605.76	*7217*	210	7635	198
flcmax_50_15_6	6290	7673	313.15	*7631*	238	7889	224
flcmax_50_15_5	6355	7679	298.54	*7496*	240	7517	225
flcmax_50_15_1	6198	7416	283.88	*7402*	240	7965	224
flcmax_50_15_8	6312	*7548*	307.38	7558	237	8143	223
flcmax_50_15_2	6531	7750	353.77	*7712*	236	8206	223
flcmax_50_20_2	6740	8838	798.83	*8836*	312	9412	297
flcmax_50_20_1	6736	8539	0.52	*8521*	312	8557	297
flcmax_50_20_7	6756	*8417*	706.38	8425	313	9369	296
flcmax_50_20_8	6897	8590	0.55	*8536*	313	8905	296
flcmax_50_20_4	6830	*8493*	1336.74	8502	312	8815	297

Results and discussion

A series of preliminary experiments was performed to investigate 20 simple constructive heuristics (SCHs). Table 1 only shows five of these heuristics owing to considerations of space (Table 3 presents the analyses of all of the SCHs). The computational results reveal that LWKR and LRM can include all of the best solutions, which are thus randomly selected in the experiments conducted in this study.

The computational times for all SCHs were below one second, implying that SCH is a quick but ineffective scheduling method.

Parameter selection may influence the quality of the results. Five different ant sizes (i.e., m=5, 10, 20, 50, 100) were tested in the preliminary experiments under the constraint of a maximum computation time, where size 5 was superior and was adopted for all further studies. All the runs were terminated after 5000 iterations. The search was still actively proceeding at each iteration, as indicate by plotting each current best solution (see Fig. 4). If solution quality is extremely important, then a better solution can be obtained through making additional computational effort. The solution presented herein is adequate for most applications. To determine the appropriate values of parameters q_0, α, β, and ρ, a preliminary optimization phase was performed on ten randomly selected instances from the above benchmark problem set. Three of these four parameters were fixed in each experiment, at values q_0=0.9, α=0.1, β=2 and ρ=0.1, and the effects of the unfixed parameter were analyzed at five levels (i.e., q_0, α, ρ, ∈{0.1,0.3,0.5,0.7, 0.9}, β∈{0.1,0.5,1,2,5}). Five trials were performed for each instance, and the best solution was selected. The computational results indicate that the best solution quality was obtained at values around α=0.1, β=2, ρ=0.1 and q_0=0.9, so these values were used for the experiments in this study. The experiments conducted herein reveal that q_0 and ρ are robust at values around 0.9 and 0.1, respectively. Additionally, the convergence speed is slower and the solution deviation of different trials rises for smaller q_0 values. In particular, the selected parameter values were found to be largely independent of the problem.

The proposed heuristic was coded in Visual C^{++} and run on a PC with Intel Pentium 4 (1.5 GHz) CPU. Each instance was tested for five trials to evaluate the heuristic. The best trial and its computation time (CPU time in seconds) were then recorded. Table 2 shows the final results.

As revealed in Table 2, the proposed MHD-ACS heuristic yielded solutions lower than the upper bounds (UB) of the benchmark problems in 32 of 40 instances. Although the solutions of the remaining eight instances are inferior, their values closely approach the UBs with an average deviation of 0.17% and a maximum deviation of 0.39%.

Notably, the computation time of the proposed heuristic is almost the same for same-size instances. Since the computation time varies with the hardware and software, this study cannot directly compare the computational efficiency. However, even for problems involving up to 1000 operations, the proposed heuristic can produce good solutions at reasonable computational costs, revealing that the proposed heuristic can be applied to real world problems.

This study also considers the case where the heuristic desirability is constant (i.e., η(i, u)=1) for all jobs (i.e., where the heuristic desirability plays no role in the algorithm). Table 2 indicates that the performance achieved by the constant heuristic desirability (CHD-ACS) method is worse than that achieved by the MHD-ACS heuristic. This finding helps evaluate the impact of the multi-heuristic desirability measure on the algorithm.

Table 3 shows the computational results of the proposed MHD-ACS heuristic and the SCHs. These data demonstrate that SCH is an ineffective scheduling method. Additionally, Table 3 indicates that the proposed MHD-ACS heuristic improved the upper bounds of the benchmark problem set, with an average deviation of –0.056% and a minimum deviation of –3.16%.

4 Conclusions

This study examines the potential of the ACS heuristic for $F||C_{\max}$. The proposed heuristic improves the upper bounds of the benchmark problems in 32 of 40 instances. Experimental results reveal that the MHD-ACS successfully solves the $F||C_{\max}$ problem. The analytical results

Table 3 Computational results of MHD-ACS and SCHs

Heuristic	Minimum deviation (%)	Maximum deviation (%)	Mean deviation (%)
MHD-ACS	*−3.16*	*0.39*	*−0.56*
SPT	43.58	89.61	65.20
LPT	60.24	105.51	86.75
LWKR	2.08	21.16	9.97
MWKR	0.54	21.05	10.24
SRM	2.08	21.16	9.97
LRM	0.54	21.05	10.24
SPT/TWK	18.75	54.68	32.70
LPT/TWK	48.58	114.15	70.03
SPT/TWKR	16.24	50.78	31.03
LPT/TWKR	47.92	89.16	68.61
SPT·TWK	14.37	73.36	28.41
LPT·TWK	46.33	110.25	67.92
SPT·TWKR	14.99	64.64	27.95
LPT·TWKR	42.01	103.72	64.72
SSO	25.05	204.18	116.20
LSO	37.66	190.66	120.07
SPT+SSO	12.95	175.01	112.89
LPT+LSO	20.58	209.46	124.42
SPT/LSO	40.54	198.02	125.43
LPT/SSO	59.53	217.88	141.36

clearly indicate that such a heuristic is worth exploring for solving different scheduling problems. Since MHD-ACS is a versatile heuristic, the heuristic proposed herein can be easily generalized to different criteria and other combinatorial optimization problems with minor modifications. Hence, we believe that ACS has significant potential for application in FSP.

This study has made a step towards establishing an efficient heuristic for the $F||C_{\max}$ problem. Future research can extend this study in several possible ways. First, the MHD-ACS provides various options and parameter settings that are worth examining in detail. Second, the developed heuristic can be extended to other manufacturing environments and to the FSP with different performance criteria. Finally, other ACO algorithms may be used on continue research into this problem.

Appendix

Simple constructive heuristics and its formulas of RI_{kl}

1. SPT: Select the job with the shortest processing time; $RI_{kl} = p_{kl}$.
2. LPT: Select the job with the longest processing time; $RI_{kl} = p_{kl}$.
3. LWKR: Select the job with the least work remaining; $RI_{kl} = \sum_{u=l}^{m} p_{ku}$.
4. MWKR: Select the job with the most work remaining; $RI_{kl} = \sum_{u=l}^{m} p_{ku}$.
5. SRM: Select the job with the shortest remaining work, excluding the operation under consideration; $RI_{kl} = \sum_{u=l+1}^{m} p_{ku}$.
6. LRM: Select the job with the longest remaining work, excluding the operation under consideration; $RI_{kl} = \sum_{u=l+1}^{m} p_{ku}$.
7. SPT/TWK: Select the job with the smallest ratio of the processing time to the total work; $RI_{kl} = p_{kl} \Big/ \sum_{u=1}^{m} p_{ku}$.
8. LPT/TWK: Select the job with the largest ratio of the processing time to the total work; $RI_{kl} = p_{kl} \Big/ \sum_{u=1}^{m} p_{ku}$.
9. SPT/TWKR: Select the job with the smallest ratio of the processing time to the total work remaining; $RI_{kl} = p_{kl} \Big/ \sum_{u=l}^{m} p_{ku}$.
10. LPT/TWKR: Select the job with the largest ratio of the processing time to the total work remaining; $RI_{kl} = p_{kl} \Big/ \sum_{u=l}^{m} p_{ku}$.
11. SPT·TWK: Select the job with the smallest value of the processing time multiplied by the total work; $RI_{kl} = p_{kl} \cdot \sum_{u=1}^{m} p_{ku}$.
12. LPT·TWK: Select the job with the largest value of the processing time multiplied by the total work; $RI_{kl} = p_{kl} \cdot \sum_{u=1}^{m} p_{ku}$.
13. SPT·TWKR: Select the job with the smallest value of the processing time multiplied by the total work remaining; $RI_{kl} = p_{kl} \cdot \sum_{u=l}^{m} p_{ku}$.
14. LPT·TWKR: Select the job with the largest value of the processing time multiplied by the total work remaining; $RI_{kl} = p_{kl} \cdot \sum_{u=l}^{m} p_{ku}$.
15. SSO: Select the job with the shortest subsequent operation; $RI_{kl} = p_{k,l+1}$.
16. LSO: Select the job with the longest subsequent operation; $RI_{kl} = p_{k,l+1}$.
17. SPT+SSO: Select the job with the smallest value of the processing time of the current plus the subsequent operation; $RI_{kl} = p_{kl} + p_{k,l+1}$.
18. LPT+LSO: Select the job with the largest value of the processing time of the current plus the subsequent operation; $RI_{kl} = p_{kl} + p_{k,l+1}$.
19. SPT/LSO: Select the job with the smallest ratio of the processing time to the subsequent operation; $RI_{kl} = p_{kl} / p_{k,l+1}$.
20. LPT/SSO: Select the job with the largest ratio of the processing time to the subsequent operation; $RI_{kl} = p_{kl} / p_{k,l+1}$.

References

1. Dudek RA, Panwalkar SS, Smith ML (1992) The lessons of flowshop scheduling research. Oper Res Soc Am 40:7–13
2. Koulamas C (1998) A new constructive heuristic for the flowshop scheduling problem. Eur J Oper Res 105:66–71
3. Reisman A, Kumar A, Motwani J (1997) Flowshop scheduling/ sequencing research: a statistical review of the literature, 1952–1994. IEEE Trans Eng Manage 44:316–329
4. King JR, Spachis AS (1980) Heuristics for flow-shop scheduling. Int J Prod Res 18:345–357
5. Park YB, Pegden CD, Enscore EE (1984) A survey and evaluation of static flowshop scheduling heuristics. Int J Prod Res 22:127–141
6. Turner S, Booth D (1987) Comparison of heuristics for flow shop sequencing. OMEGA 15:75–85
7. Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1:117–129
8. Widmer M, Hertz A (1989) A new heuristic method for the flow shop sequencing problem. Eur J Oper Res 41:186–193
9. Ogbu FA, Smith DK (1990) The application of the simulated annealing algorithm to the solution of the n / m / $C_{\max}$ flowshop problem. Comput Oper Res 17:243–253
10. Chen CL, Vempati VS, Aljaber N (1995) An application of genetic algorithms for flow shop problems. Eur J Oper Res 80:389–396
11. Potts CN, Van Wassenhove LN (1991) Single machine tardiness sequencing heuristics. IIE Trans 23:346–354

12. Dorigo M, Di Caro G, Gambardella LM (1999) Ant algorithms for discrete optimization. Arti Life 5:137–172
13. Colorni A, Dorigo M, Maniezzo V (1991) Distributed optimization by ants colonies. In: Verela F, Bourgine P (eds) Proceedings of the first European Conference on Artificial Life (ECAL'91). MIT Press, Cambridge, Mass, USA, pp 134–142
14. Dorigo M (1992) Optimization, learning and natural algorithms. PhD Thesis, DEI, Politecnico di Milano, Italy
15. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evolu Comput 1:53–66
16. Gambardella LM, Dorigo M (1997) HAS-SOP: hybrid ant system for the sequential ordering problem. Technical Report IDSIA-11-97, IDSIA, Lugano, Switzerland
17. Gambardella LM, Taillard E, Dorigo M (1999) Ant colonies for the quadratic assignment problem. J Oper Res Soc 50:167–176
18. Bullnheimer B, Hartl RF, Strauss C (1999) An improved ant system algorithm for the vehicle routing problem. Ann Oper Res 89:319–334
19. Costa D, Hertz A (1997) Ants can colour graphs. J Oper Res Soc 48:295–305
20. Kuntz P, Layzell P, Snyers D (1997) A colony of ant-like agents for partitioning in VLSI technology. Proceedings of the Fourth European Conference on Artificial Life. The MIT Press, Cambridge, MA
21. Song YH, Chou CS (1998) Application of ant colony search algorithms in power system optimization. IEEE Power Eng Rev 18:63–64
22. Di Caro G, Dorigo M (1998) AntNet: distributed stigmergetic control for communications networks. J Arti Intel Res 9:317–365
23. Gagnë C, Price WL, Gravel M (2002) Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. J Oper Res Soc 53:895–906
24. Ying KC, Liao CJ (2003) An ant colony system approach for scheduling problem. Prod Plan Cont 14:68–75
25. Rajendran C, Ziegler H (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. Eur J Oper Res 155:426–438
26. Ying KC, Liao CJ (2004) An ant colony system for permutation flow-shop sequencing. Comput Oper Res 31:791–801
27. Colorni A, Dorigo M, Maniezzo V, Trubian M (1994) Ant system for job-shop scheduling. Belgian J Oper Res Stat Comput Sci (JORBEL) 34:39–53
28. Christian B (2005) Beam-ACO¯hybridizing ant colony optimization with beam search: an application to openshop scheduling. Comput Oper Res 32:1565–1591
29. Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discete Math 5:287–362
30. James ED, Michael PH (1970) Review of sequencing research. Nav Res Log Q 17:11–39
31. Haupt R (1989) A survey of priority rule-based scheduling. OR Spek 11:3–16
32. Chang YL, Toshiyuki S, Robert SS (1996) Ranking dispatching rules by data envelopment analysis in a job shop environment. IIE Trans 28:631–642
33. Holthaus O, Rajendran C (2000) Efficient jobshop dispatching rules: further developments. Prod Plan Cont 11:171–178
34. Demirkol E, Mehta S, Uzsoy R (1998) Benchmarks for shop scheduling problems. Euro J Oper Res 109:137–141
35. Pinedo M (1995) Scheduling: theory, algorithms and system. Prentice-Hall, New Jersey