Int J Adv Manuf Technol (2012) 60:1139–1147
DOI 10.1007/s00170-011-3671-1

ORIGINAL ARTICLE

A hybrid EDA with ACS for solving permutation flow shop scheduling

Yeu-Ruey Tzeng · Chun-Lung Chen · Chuen-Lung Chen

Received: 19 April 2011 / Accepted: 26 September 2011 / Published online: 12 October 2011

Abstract This paper proposes a hybrid estimation of distribution algorithm (EDA) with ant colony system (ACS) for the minimization of makespan in permutation flow shop scheduling problems. The core idea of EDA is that in each iteration, a probability model is estimated based on selected members in the iteration along with a sampling method applied to generate members from the probability model for the next iteration. The proposed algorithm, in each iteration, applies a new filter strategy and a local search method to update the local best solution and, based on the local best solution, generates pheromone trails (a probability model) using a new pheromone-generating rule and applies a solution construction method of ACS to generate members for the next iteration. In addition, a new jump strategy is developed to help the search escape if the search becomes trapped at a local optimum. Computational experiments on Taillard's benchmark data sets demonstrate that the proposed algorithm generated high-quality solutions by comparing with the existing population-based search algorithms, such as genetic algorithms, ant colony optimization, and particle swarm optimization.

Y.-R. Tzeng · C.-L. Chen
Department of MIS,
National Chengchi University,
No. 64, Sec. 2, ZhiNan Rd., Wenshan District,
Taipei City 11605 Taiwan, Republic of China

Y.-R. Tzeng
e-mail: 93356511@nccu.edu.tw

C.-L. Chen
e-mail: chencl@mis.nccu.edu.tw

C.-L. Chen (✉)
Department of Accounting Information,
Takming University of Science and Technology,
No. 56, Sec. 1, Huanshan Road, Neihu District,
Taipei City, Taiwan, Republic of China
e-mail: charleschen@takming.edu.tw

Keywords Ant colony system · Estimation of distribution algorithm · Permutation flow shop scheduling · Makespan

1 Introduction

This paper proposes a hybrid estimation of distribution algorithm (EDA) with ant colony system (ACS), denoted as EDA_{ACS}, for the minimization of makespan in permutation flow shop scheduling problems (PFSP-makespan). The candidate problem determines the best sequence of n jobs that are to be processed on m machines in the same order in order to minimize the complete time of the last job on the last machine (makespan). It has been proven to be one of the most studied NP-complete scheduling problems [1]. Therefore, the development of approximate algorithms has held the attention of many researchers in recent decades.

Both EDA, proposed by Mühlenbein and Paaß [2], and ACS, proposed by Dorigo and Gambardella [3], are population- and stochastic-based metaheuristics. The core idea of EDA is that in each iteration, it examines the members in the iteration to gain insight information of the members and exploits the information to generate new members for the next iteration afterwards. A probability distribution model is estimated based on selected members in the iteration, and a sampling method is applied to generate members from the probability distribution model for the next iteration. Santana et al. [4] pointed out that EDA is able to overcome some drawbacks of traditional genetic algorithms (GA). A few EDA algorithms have been developed for solving scheduling problems such as the project scheduling problem [5] and the permutation flow shop scheduling problem [6].

The ACS was developed for solving combinatorial optimization problems using principles of communicative behavior found in real ant colonies. In general, the ACS approach solves

an optimization problem by iterating the following two steps [7]: constructing members in an iteration using initial pheromone trails and modifying the pheromone trails using the members in a way that is deemed to bias future sampling toward high-quality solutions. The major difference between EDA and ACS is that in EDA, a probability model generated in an iteration is estimated based on only the selected members in the iteration; however, in ACS, the pheromone trails are cumulated through the whole search process. The ACS algorithm has been successfully applied to solve several scheduling problems such as the single-machine scheduling problem [8], the parallel processor scheduling problem [9], the permutation flow shop scheduling problem [10–14], the job shop scheduling problem [15], and the joint production and maintenance scheduling problem [16]. Most of the research showed that ACS outperformed genetic algorithms, simulated annealing, and tabu search.

The proposed hybrid algorithm, EDA_{ACS}, integrates the ideas of ACS into EDA. In each iteration of the searching process, it applies a new filter strategy and a local search method to update the local best solution; then, based on the local best solution, it generates pheromone trails (a probability model) and applies a solution construction method of ACS, according to the pheromone trails, to generate members for the next iteration. In addition, a new jump strategy is developed to help the search escape if the search becomes trapped at a local optimum. Computational experiments on Taillard's benchmark data sets will be performed to evaluate the effectiveness of the proposed algorithm by comparing its performance with population-based search algorithms, such as genetic algorithms, ant colony optimization, and particle swarm optimization.

The remainder of the paper is organized as follows: Section 2 gives the problem statement. The proposed EDA_{ACS} is described in Section 3. Section 4 provides computational experiments, and the conclusion is presented in Section 5.

2 Problem statement: PFSP-makespan

PFSP-makespan can be denoted as $Fm|prmu|C_{\max}$ using the notation proposed by Graham et al. [17]; given a set J of n jobs, a set M of m machines, and processing times p_{ij} for each job j on each machine i, the problem consists of scheduling all n jobs at each one of the m machines. The processing sequence of the jobs must be the same on all the machines, and each job j can only start its execution on a machine i if both the previous job on the same machine i and the same job j on the previous machine $i-1$ have already been processed. Furthermore, the order in which a job must pass through the machines is predefined and identical for all the jobs. The objective of this problem is to determine a job ordering that minimizes the completion time of the last job in the last machine, called the makespan. Although Garey et al. [1] showed that the problem with two machines can be solved in polynomial time, the general case with m machines is known to be NP-hard. Given a permutation schedule $j_1,\ldots, j_n$ for an m machine flow shop, the completion time of job j_k at machine i, C_{i,j_k}, can be computed easily through a set of recursive equations:

$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \qquad i = 1,2,\ldots,m \tag{1}$$

$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_l} \qquad k = 1,2,\ldots,n \tag{2}$$

$$C_{i,j_k} = \max(C_{i-1,j_k}, C_{i,j_{k-1}}) + p_{i,j_k} \qquad i = 2,\ldots,m;\ \mathrm{k} = 2,\ldots,n \tag{3}$$

Then makespan, $C_{\max}$, is obtained by $C_{\max} = C_{m,j_n}$.

3 The proposed algorithm: EDA_{ACS}

The EDA_{ACS} algorithm follows the searching process of EDA. The main steps of the process of EDA are presented as follows [4]:

1. Set iteration t=0. Generate M solutions randomly.
2. **do** {
3. Evaluate the solutions using the fitness function.
4. Select a set D of $N \leqq M$ solutions according to a selection method.
5. Generate a probability model based on the solutions in D.
6. Generate M new solutions sampling from the probability model generated in step 5.
7. t=t+1.
8. } **until** Terminate criteria are met.

The EDA_{ACS} algorithm integrates the ideas of ACS into the process of EDA by modifying the initial step and the loop in EDA. It modifies the initial step by producing only a solution using the heuristic, NEHT [18], and then setting the solution to be the local best solution. The loop is modified by generating pheromone trails based on the local best solution, constructing M new solutions according to the pheromone trails by applying a solution construction method, and updating the local best solution with the solution produced by applying a new filter strategy and a local search method to the M new solutions. Note that since the solution construction method is revised based on the probabilistic action rule [3] used in ACS, we see a solution constructed in EDA_{ACS} as a solution constructed by an artificial ant and denote the number of solutions, M, constructed in an

iteration as *Num-Ant* hereafter. In addition, since a new jump strategy is developed in EDA_{ACS} to help the search escape if the search becomes trapped at a local optimum, the jump strategy is included in EDA_{ACS}. The searching process of EDA_{ACS} is presented below, and the major ideas of EDA_{ACS}, pheromone-generating rule, solution construction method, filter strategy, and jump strategy are discussed in detail in the following sections.

1. Set t=0. Generate a solution using NEHT and let it be the local best solution.
2. **do** {
3. Generate pheromone trails based on the local best solution.
4. Generate *Num-Ant* new solutions according to the pheromone trails generated in step 3.
5. Apply the filter strategy and a local search method to select a solution from the *Num-Ant* solutions generated in step 4 and update the local best solution with the selected solution.
6. Launch the jump strategy if the search traps into a local optimum.
7. t=t+1.
8. } **until** Terminate criteria are met.

3.1 Pheromone-generating rule and solution construction method

The new pheromone-generating rule is applied to generate pheromone trails when the local best solution is updated in every iteration. The following example illustrates the procedure of the generating rule. Given that the updated local best solution in an iteration is Π'=(3, 1, 2, 5, 4), and let $\tau(i, u)$ denote the pheromone value of job u on position i, the generating rule first assigns a pheromone value, τ_l, to each job on its position in Π'; that is, $\tau(1, 3)=\tau(2, 1)=\tau(3, 2)=\tau(4, 5)=\tau(5, 4)=\tau_l$. Then, for each job, the new rule assigns a pheromone value, τ_p, to the positions prior to its position in Π' and assigns a pheromone value, τ_s, to the positions succeeding its position in Π'. For instance, job 2 is on position 3 in Π', so $\tau(1, 2)=\tau(2, 2)=\tau_p$ and $\tau(4, 2)=\tau(5, 2)=\tau_s$. These three pheromone values, τ_l, τ_p, and τ_s, have to be properly determined as to allow the pheromone trails to keep the sequence of the jobs in the local best solution while constructing new solutions, and the valuable information of the sequence of the jobs can be retained in every iteration. The following solution construction method will clearly illustrate this idea.

A solution construction of an artificial ant is composed of job selections from the first position to the last position for the solution. A revised job selection rule based on the probabilistic action rule of Dorigo and Gambardella [3] is proposed in EDA_{ACS}. Given a parameter value, q_0 ($0 \le q_0 \le 1$), an artificial ant, ant k, first generates a random number, q, from a uniform distribution ranged [0, 1]. If q is less than or equal to q_0, then Eq. 4 is used to select a job; otherwise, a probabilistic action rule Eq. 5 is applied to select a job.

$$j = \arg\max_{u \in S_k(i)}\{\tau(i,u)\}, \qquad \text{if } q \le q_0 \tag{4}$$

$$P_k(i,j) = \frac{\tau(i,j)}{\sum_{u \in S_k(i)} \tau(i,u)}, \qquad \text{if } q > q_0 \tag{5}$$

where $S_k(i)$ is the set of unscheduled jobs of ant k positioned on job i.

To better understand the solution construction method, the previous local best solution, Π'=(3, 1, 2, 5, 4), is used, and let τ_l=100, τ_p=1, and τ_s=110. Table 1 summarizes the pheromone values for all the jobs on different positions. The solution construction method for this example is presented as follows. Job selection for position 1: if $q \le q_0$, since $S_k(i)$={1, 2, 3, 4, 5}, and $\tau(1, 3)=\tau_l$=100 and $\tau(1, 1)=\tau(1, 2)=\tau(1, 4)=\tau(1, 5)=\tau_p$=1, job 3 ($j = \arg\max_{u \in S_k(i)}\{\tau(i,u)\} = 3$) is selected for position 1; if $q > q_0$, each job j will be selected with a probability of $\tau(1, j)/(100+4\times1)$, respectively, and a random number, between 0 and 1, is generated to select a job for position 1. Job selection for position 2 under the condition that job 5, not job 3, is selected for position 1: if $q \le q_0$, since $S_k(i)$={1, 2, 3, 4}, and $\tau(2, 3)$=110, $\tau(2, 1)$=100, and $\tau(2, 2)=\tau(2, 4)$=1, job 3 ($j = \arg\max_{u \in S_k(i)}\{\tau(i,u)\} = 3$) is selected for position 2; if $q > q_0$, then job 3 has the highest probability, $\tau(2, 3)/(110+100+1+1)=110/(212)$, to be selected for position 2. This result shows that if job 3 is not selected for position 1, its position in Π', it will be selected for position 2 with the highest probability. This illustrates the idea of the new pheromone-generating rule which will keep a job on its position in the local best solution while constructing new solutions.

The simple example also shows that a high q_0 value will cause the job selection rule to highly retain the job sequence in the local best solution and cause premature

Table 1 Example illustrating the pheromone-generating rule and the solution construction rule

Position	Job				
	3	1	2	5	4
1	110	1	1	1	1
2	110	100	1	1	1
3	110	110	100	1	1
4	110	110	110	100	1
5	110	110	110	110	100

Table 2 Experimental parameters

Parameters	Levels	Total levels
q_{high}–q_{low}	0.98–0.68, 0.96–066, 0.92–0.62, 0.88–0.58, 0.84–0.54, 0.8–0.5	6
f-size	0, 1, 4, 9, 14, 18	6
Jump-rate	0, 1.03, 1.06, 1.09, 1.12, 1.15, 1.18, none	8
	Total parameter combinations	288

convergence. In order to mitigate this problem, a variable q_0 setting method is applied in EDA_{ACS}. We let artificial ants use different q_0 values to construct feasible solutions. Given that there are *Num_Ant* artificial ants considered in a population, artificial ant k is the kth artificial ant, and q_0 varies from q_{high} to q_{low}, the q_0 for artificial ant k is calculated as follows:

$$q_{0,k} = q_{high} - \left[\left(q_{high} - q_{low}\right) \times k/Num\_Ant\right].$$

For example, if we set ant size, *Num_Ant*, to be 10, k is between 0 and 9, and q_0 varies from 0.96 (q_{high}) to 0.66 (q_{low}), the set of $q_{0,k}$ is {0.96, 0.93, 0.90,…, 0.69}. Note that the higher the $q_{0,k}$ value, the higher the exploitative capability the ant has, and the lower the $q_{0,k}$ value, the higher the explorative capability the ant has. Therefore, including the ants with different q_0 values in a population may balance the exploitative capability and the explorative capability while searching the solution space.

3.2 Filter strategy and local search method

The proposed filter strategy and a local search method, NEHT_LS, are applied to update the local best solution when all the artificial ants (*Num_Ant*) finish constructing their solutions in an iteration. We define filter list as a first-in, first-out queue to store the makespan of the chosen solution in each iteration and set a parameter called filter size, *f-size*, to define the size of the queue. The queue is set to be empty initially. When all the *Num_Ant* solutions are constructed, the solutions are sorted according to their makespans ascendingly and the filter strategy is applied from the top of the *Num_Ant* solutions until the first solution, whose makespan is different from all the makespans in the filter list, is found and store the makespan of the solution in the filter list. If none of the *Num_Ant* solutions has a different makespan from the makespans in the filter list, the last of the *Num_Ant* solutions is chosen. The purpose of comparing makespans, instead of job sequences, of solutions while using the filter strategy is twofold. Firstly, it may guide the search to the solution regions which have not been examined and, secondly, it can significantly reduce the computation time while comparing the solution constructed by an artificial ant and the solutions stored in the filter list; this is especially critical when the number of jobs considered in a problem is large. In addition, the idea of choosing the solution with the largest makespan, when none of the *Num_Ant* solutions has a different makespan from the makespans in the filter list, is that it may keep the search of EDA_{ACS} from quick convergence.

Once a solution is chosen using the filter strategy, the local search method, NEHT_LS, is applied to improve the makespan of the solution. NEHT_LS integrates Taillard's [19] modified-NEH method with Ruiz and Stutzle's [20] iterative improvement method. Given that Π is the job sequence of the chosen solution, NEHT_LS first randomly chooses a job k and removes it from Π, and then inserts job k into the first position, the last position, and the positions between every two consecutive jobs in Π to generate n different solutions; let Π'' be the best of the n generated solutions. If the makespan of Π'' is smaller than that of Π, NEHT_LS will update Π with Π'' and repeat the same procedure until Π cannot be further improved. If the makespan of Π is smaller than that of the local best solution, update the local best solution with Π.

3.3 Jump strategy

The main idea of the jump strategy is to guide the search to jump to another solution region when the search is trapped

Table 3 ANOVA table for testing the significance of the three parameters

Source of variance	Sum of squares	*df*	Mean squares	*F*	Significance
V1 (q_{high}–q_{low})	0.049	5	0.01	0.014	1.000
V2 (*f-size*)	154.917	5	30.983	44.400	0.00[a]
V3 (*Jump-rate*)	6.395	7	0.914	1.309	0.241
V1 × V2	3.461	25	0.138	0.198	1.000
V1 × V3	0.531	35	0.015	0.022	1.000
V2 × V3	15.966	35	0.456	0.654	0.942
V1 × V2 × V3	2.245	175	0.013	0.018	1.000
Error	2,210.692	3,168	0.698		
Total	2,394.256	3,455			

[a]Difference in the effects at a significance level of 0.01

Table 4 Results of Duncan's test for different filter sizes

f-size	Average ARP	Subsets	
		1	2
18	0.6436	A	
14	0.6493	A	
9	0.6499	A	
4	0.6614	A	
1	0.6743	A	
None	1.2232		B

in a local optimum. We define the search trapped in a local optimum when the search is not able to improve the best-so-far solution in a number of iterations. The solution generated by the jump strategy is considered to be a new initial solution, and the search procedure is restarted.

Two jumping distances, objective value distance and sequence structure distance, are used in this study. Objective value distance implies that a threshold value is set to guarantee that a jump is far enough from the current local best solution. We set a parameter, *Jump-rate*, to calculate the objective value distance, objective value distance=*Jump-rate*×objective value of the current local best solution. When a local optimum is detected, an objective value distance is calculated and the makespans of the solutions constructed by the *Num_Ant* artificial ants in the current iteration compared with the objective value distance. Only the solutions that have makespans larger than the objective value distance are considered to be the candidates for a new initial solution. If none of the *Num_Ant* solutions has makespan larger than the objective value distance, randomly choose a solution from the *Num_Ant* solutions and use it as the new initial solution. If there is more than one candidate, a sequence structure distance is applied to select a suitable one. A sequence structure distance measures the structure similarity between two job sequences, S_1 and S_2. Let (i, u_1) be the job on position i in S_1 and (i, u_2) be the job on position i in S_2; define the distance between S_1 and S_2 on position i, $d(i, u)$ as 0 if $u_1=u_2$ and as 1 if $u1\neq u_2$. The sequence structure distance between S_1 and S_2 is then defined to be the sum of $d(i, j)$ for all the positions.

Table 6 Computational results of M-MMAS, PACO, and EDA_{ACS} (t_{30})

Test problems	M-MMAS	PACO	EDA_{ACS}
20×5	0.11	0.2	0.04
20×10	0.15	0.32	0.07
20×20	0.09	0.31	0.10
50×5	0.02	0.08	0.02
50×10	1.3	0.9	0.75
50×20	2.1	1.46	1.25
100×5	0.03	0.04	0.06
100×10	0.46	0.35	0.39
100×20	2.59	2.17	1.55
200×10	0.37	0.26	0.24
200×20	2.34	2	1.58
500×20	1.06	0.98	0.81
Average	0.885	0.756	0.572

$t_{30}=n\times(m/2)\times 30$ ms

4 Computational experiments

The well-known Taillard's test problems for PFSP-makespan [21] are used to evaluate the performance of EDA_{ACS}. The test problems are composed of 12 different problem sets with different numbers of jobs and different numbers of machines. Twelve instances, selecting the first instance from each of the 12 problem sets, denoted as $Test_1$, are used to investigate the effects of the three major parameters of EDA_{ACS}: the range of q_0 (q_{high}–q_{low}), the filter size (*f-size*), and the *Jump-rate*. Afterwards, the EDA_{ACS} with the best combination of the three parameters are applied to solve all the test problems in order to evaluate its performance. Note that all the algorithms in this research are coded in C language and executed on the Linux operating system.

The levels considered for the three major parameters for EDA_{ACS} are summarized in Table 2. Six levels are set for q_{high}–q_{low}: 0.98–0.68, 0.96–066, 0.92–0.62, 0.88–0.58, 0.84–0.54, 0.8–0.5; six levels are set for *f-size*: none, 1, 4, 9, 14, and 18, where none refers to no filter strategy being applied; and eight levels are set for *Jump-rate*: none, 0.0, 1.03, 1.06, 1.09, 1.12, 1.15, and 1.18, where none refers to

Table 5 Response table for the three parameters

	Level of the parameters						Difference
q_{high}–q_{low}	0.98–0.68	0.96–066	0.92–0.62	0.88–0.58	0.84–0.54	0.8–0.5	
	0.6300	0.6411	0.6468	0.6557	0.6703	0.6685	0.0403
f-size	1	4	9	14	18		
	0.6717	0.6578	0.6463	0.6444	**0.6401**		0.0316
Jump-rate	1.03	1.06	1.09	1.12	1.15	1.18	
	0.6785	0.6583	0.6389	**0.6359**	0.6389	0.6364	0.0426

Table 7 Computational results of M-MMAS, PACO, and EDA_{ACS} (t_{60})

Test problems	M-MMAS	PACO	EDA_{ACS}
20×5	0.08	0.08	0.00
20×10	0.09	0.09	0.07
20×20	0.07	0.07	0.06
50×5	0.02	0.02	0.01
50×10	1.14	1.14	0.74
50×20	2.06	2.06	1.1
100×5	0.02	0.02	0.04
100×10	0.42	0.42	0.28
100×20	2.5	2.5	1.45
200×10	0.32	0.32	0.18
200×20	2.18	2.18	1.43
500×20	1.09	1.09	0.73
Average	0.833	0.833	0.508

$t_{60}=n\times(m/2)\times 60$ ms

no jump strategy being applied and 0.0 refers to the condition that only sequence structure distance is considered. The remaining parameters of EDA_{ACS} are described as follows: the size of the artificial ants (*Num_Ant*) used in each iteration is set to be 10; the τ values used in the new pheromone-generating rule, τ_p, τ_l, and τ_s, are set to be 1, 950, and 1,000 respectively; and the number of iterations without improvement for defining trapping at a local optimum is set to be the number of machines of the instances solved. All these parameters are determined by trial and error. Therefore, there are a total of 288 different combinations of the three parameters. The EDA_{ACS} is then applied with each of the 288 combinations to solve the 12 instances in $Test_1$ with limited computation times, $n\times(m/2)\times 30$ ms [22], for three trials, where n refers to the number of jobs and m refers to the number of machines for the instances. The performance of the EDA_{ACS} with a combination of the three parameters for an instance is evaluated using average relative performance (ARP): $\mathrm{ARP}=\sum_{i=1}^{R}\left(\frac{Heu_i-Best_{sol}}{Best_{sol}}\times 100\right)/R$ where Heu_i is the makespan obtained by any of the three trials of the EDA_{ACS} with the combination of the parameters and $Best_{sol}$ is the best makespan that all the research has found for the instance provided by Zobolas et al. [23].

Analysis of variance (ANOVA) is applied to analyze the ARPs produced by EDA_{ACS} with all the 288 combinations. Table 3 presents the results of ANOVA. The results show that the filter strategy significantly affects the ARP of the test problems. Therefore, the Duncan's test is applied to test whether the performance of any two levels of the filter strategy is significantly different. Table 4 presents the results of the Duncan's test. The results show that the major difference is seen from the level "none" and the other levels. In addition, although jump strategy is not significant in the analysis of variance, it is found, from the ARPs, that EDA_{ACS} with jump strategy always outperforms the EDA_{ACS} without jump strategy; the average ARP of the EDA_{ACS} with jump strategy is about 12% ((0.827 − 0.724)/0.827) better than that of the EDA_{ACS} without jump strategy.

ANOVA is then applied to analyze the ARPs under the condition that the filter strategy and the jump strategy are applied in EDA_{ACS}. The results show that none of the three parameters are significant. A commonly used tool in experimental design [24], the response table analysis, is further applied to investigate the effect of the parameters. Table 5, a response table, summarizes the average ARP for each level of the three parameters. The minimum average ARP for each parameter is shown in bold, and the difference between the maximum average ARP and the minimum average ARP for each parameter is presented in the last column. The minimum average ARP for each parameter is: q_{high}–q_{low}=0.98–0.68, *f-size*=18, and *Jump-rate*=1.12; the condition is very close to the condition that generates the best solution: q_{high}–q_{low}=0.98–0.68, *f-size*=14, and *Jump-rate*=1.12. Since the difference between the average ARP of *f-size*=14 (0.6444) and the average ARP of *f-size*=18 (0.6401) is negligible, the best parameter combination for EDA_{ACS} is q_{high}–q_{low}=0.98–0.68, *f-size*=14, and *Jump-rate*=1.12. Furthermore, the EDA_{ACS} is applied to the same test problems under the condition fixed q_0=0.98, *f-size*=14, and *Jump-rate*=1.12 in order to evaluate the effect of variable q_0. Computational results show that the average ARP produced by the EDA_{ACS} using fixed q_0 is 0.639, which is about 9% ((0.639 − 0.579)/0.639) worse than the average ARP produced by the EDA_{ACS} using

Table 8 Computational results of M-MMAS, PACO, and EDA_{ACS} (t_{90})

Test problems	M-MMAS	PACO	EDA_{ACS}
20×5	0.04	0.18	0.00
20×10	0.07	0.24	0.04
20×20	0.06	0.18	0.04
50×5	0.02	0.05	0
50×10	1.08	0.81	0.63
50×20	1.93	1.41	1.01
100×5	0.02	0.02	0.04
100×10	0.39	0.29	0.24
100×20	2.42	1.93	1.3
200×10	0.3	0.23	0.18
200×20	2.15	1.82	1.39
500×20	1.02	0.85	0.69
Average	0.792	0.668	0.463

$t_{90}=n\times(m/2)\times 90$ ms

Table 9 Results of the paired samples t test for M-MMAS, PACO, and EDA_{ACS} under different computation times

Time	Algorithm	Paired differences				t	Significance
		Mean	SEM	95% Confidence interval			
				Lower	Upper		
t_{30}[a]	M-MMAS–EDA_{ACS}	0.31000	0.11091	0.06590	0.55410	2.795	0.017
	PACO–EDA_{ACS}	0.18083	0.05442	0.06106	0.30061	3.323	0.007
t_{60}	M-MMAS–EDA_{ACS}	0.32500	0.11198	0.07853	0.57147	2.902	0.014
	PACO–EDA_{ACS}	0.18250	0.04838	0.07601	0.28899	3.772	0.003
t_{90}	M-MMAS–EDA_{ACS}	0.32833	0.11471	0.07587	0.58080	2.862	0.015
	PACO–EDA_{ACS}	0.20417	0.05501	0.08308	0.32525	3.711	0.003

[a] $t_{30}=n\times(m/2)\times 30$ ms, $t_{60}=n\times(m/2)\times 60$ ms, $t_{90}=n\times(m/2)\times 90$ ms

variable q_0 (q_{high}–q_{low}=0.98–0.68). This illustrates that using different q_0 values for the artificial ants to construct solutions in an iteration is able to improve the explorative capability for EDA_{ACS}.

The EDA_{ACS} with the best parameter combination is then applied to solve the test problems in all the problem sets, and its performance is first compared with the two best ACO algorithms, M-MMAS and PACO [25], for PFSP-makespan and then with three hybrid metaheuristics, NEGAvns, HGA_RMA, PSOvns [23], which reported very promising solutions for PFSP-makespan.

Ruiz et al. [22] compared the performance of M-MMAS and PACO with other heuristics based on the same number of replication runs (R=5) and the same computation times: $n\times(m/2)\times 30$, $n\times(m/2)\times 60$, and $n\times(m/2)\times 90$ ms. All the algorithms were run on a PC with Intel Pentium IV at 2.8 GHz. Therefore, we compared the performance of EDA_{ACS} with M-MMAS and PACO based on the same computation times using a PC with the same computing power. Tables 6, 7, and 8 presents the average ARPs produced by M-MMAS, PACO, and EDA_{ACS} for the 12 problem sets with each of the three computation times, respectively. The paired samples t test is applied to test whether the performance of EDA_{ACS} significantly dominates M-MMAS and PACO, respectively. Table 9 summarizes the results of all the paired samples t tests. The results show that EDA_{ACS} significantly dominates M-MMAS and PACO under all the different computation times.

Table 10 presents the average ARPs generated by NEGAvns, HGA_RMA, PSOvns, and EDA_{ACS} on a PC with Intel Pentium IV at 2.4 GHz under the same computation time, $n\times m/10$ s, and the same number of replication runs (R=10) [23]. The paired samples t test is applied to compare the performance between EDA_{ACS} and each of the following algorithms: NEGAvns, HGA_RMA, and PSOvns. These tests show that EDA_{ACS} does not significantly dominate any of NEGAvns, HGA_RMA, and PSOvns. However, the results show that EDA_{ACS} is superior to $NEGA_{VNS}$ in 6 out of the 12 problem sets and ties in 3 out of the 12 problem sets. Overall, EDA_{ACS} dominates $NEGA_{VNS}$ by 9% ((0.466 − 0.424)/0.466). EDA_{ACS} outperforms HGA_RMA in 8 out of the 12 problem sets and ties in 1 out of the 12 problem sets. EDA_{ACS} dominates HGA_RMA by 5%. Also, EDA_{ACS} outperforms PSOvns in 7 out of the 11 problem sets and ties in 1 out of the 11 problem sets; EDA_{ACS} dominates PSOvns by 15%.

5 Conclusions

We have developed a hybrid EDA with ACS (EDA_{ACS}) to solve a permutation flow shop scheduling problem, PFSP-makespan. The EDA_{ACS}, in each iteration, applies a new filter strategy and a local search method to update the local best solution and, based on the local best solution, generates pheromone trails using a new pheromone-

Table 10 Computational results of HGA_RMA, PSOvns, NEGAvns, and EDA_{ACS}

Test problems	NEGAvns	HGA_RMA	PSOvns	EDA_{ACS}
20×5	0	0.04	0.03	0
20×10	0.01	0.02	0.02	0.01
20×20	0.02	0.05	0.05	0.01
50×5	0	0	0	0
50×10	0.82	0.72	0.57	0.6
50×20	1.08	0.99	1.36	0.9
100×5	0	0.01	0	0.04
100×10	0.14	0.16	0.18	0.21
100×20	1.4	1.3	1.45	1.25
200×10	0.16	0.14	0.18	0.12
200×20	1.25	1.26	1.35	1.3
500×20	0.71	0.69	–[a]	0.65
Average	0.466	0.448	0.472	0.424

[a] The authors did not provide results for the 500×20 instance group

generating rule and applies a solution construction method of ACS to generate members for the next iteration. Computational experiments on Taillard's test problems showed that the filter strategy significantly affects the performance of EDA_{ACS}. This result reveals that the filter strategy is able to provide valuable information in every iteration. By applying the information, the pheromone-generating rule and the solution construction method can guide the search to promising solution regions. In addition, although the ANOVA showed that the jump strategy does significantly affect the performance of EDA_{ACS}, it was found that the EDA_{ACS} with jump strategy dominates the EDA_{ACS} without jump strategy by about 12% on the average ARP. Furthermore, it was found that under the condition with the best parameter combination, the EDA_{ACS} using variable q_0 (q_{high}–q_{low}=0.98–0.68) dominates the EDA_{ACS} using fixed q_0 (q_0=0.98) by about 9%.

Computational results also showed that EDA_{ACS} with the best parameter combination outperforms several effective population-based metaheuristics. The paired t test showed that the EDA_{ACS} significantly outperforms M-MMAS and PACO, two best ACO algorithms for PFSP-makespan. Although ANOVA showed that the EDA_{ACS} does not significantly outperform NEGAvns, HGA_RMA, PSOvns, the average ARP of the EDA_{ACS} dominates that of NEGAvns, HGA_RMA, and PSOvns by 9%, 5%, and 15%, respectively. These results conclude that the EDA_{ACS} is an effective and efficient algorithm for PFSP-makespan.

A couple of ideas deserve to be further studied. First, as mentioned above, the τ values, τ_l, τ_p, and τ_s, of the pheromone-generating rule and the range for the variable q_0 may affect the exploitative capability and the explorative capability of EDA_{ACS}. A more thorough investigation on the interaction effect of these two factors may be able to find the appropriate relationship between the two factors and the exploitative capability and the explorative capability of EDA_{ACS}. Therefore, the performance of EDA_{ACS} may be improved. Second, due to the limited computation times, the smallest structure distance was chosen in the jump strategy in EDA_{ACS}. If the limit of computation time is relaxed, a study on the relationship between the structure distance and the effect of the jump strategy on the performance of EDA_{ACS} is deserved. In addition, the EDA_{ACS} algorithm can be extended to solve other permutation scheduling problems.

Acknowledgements This paper was supported in part by the National Science Council, Taiwan, ROC, under the contract NSC 100-2221-E-004 -004. The authors are grateful to the anonymous referees for their constructive comments that have greatly improved the presentation of this paper.

References

1. Garey MR, Johnson DS, Sethi R (1976) The complexity of flow shop and job shop scheduling. Math Oper Res 1(2):117–129
2. Mühlenbein H, Paaß G (1996) From recombination of genes to the estimation of distributions I. Binary parameters. In: Ebeling W, Rechenberg I, Voight H-M et al (eds) Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature—PPSN 4. Springer, Heidelberg, pp 178–187
3. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the travelling salesman problem. IEEE T Evolut Comput 1:53–66
4. Santana R, Larrañaga P, Lozano JA (2009) Research topics in discrete estimation of distribution algorithms. Memetic Comput 1 (1):35–54
5. Wang L, Fang C (2012) An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. Comput Oper Res 39(2):449–460
6. Jarboui B, Eddaly M, Siarry P (2009) An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Comput Oper Res 36(9):2638–2646
7. Blum C (2005) Ant colony optimization: introduction and recent trends. Phys Life Rev 2(4):353–373
8. Liao CJ, Juan HC (2007) An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Comput Oper Res 34(7):1899–1909
9. Srinivasa Raghavan NR, Venkataramana M (2009) Parallel processor scheduling for minimizing total weighted tardiness using ant colony optimization. Int J Adv Manuf Technol 41:986–996
10. Stützle T (1998) An ant approach to the flow shop problem. Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing, vol 3, Aachen, Germany, pp 1560–1564
11. Ying KC, Liao CJ (2004) An ant colony system for permutation flow-shop sequencing. Comput Oper Res 31(5):791–801
12. Mirabi M (2011) Ant colony optimization technique for the sequence-dependent flowshop scheduling problem. Int J Adv Manuf Technol 55(1–4):317–326
13. Ying KC, Lin SW (2006) Multiprocessor task scheduling multistage hybrid flow-shops: an ant colony system approach. Int J Prod Res 44:3161–3177
14. Alaykýran K, Engin O, Döyen A (2007) Using ant colony optimization to solve hybrid flow shop scheduling problems. Int J Adv Manuf Technol 35:541–550
15. Huang RH, Yang CL (2008) Ant colony system for job shop scheduling with time windows. Int J Adv Manuf Technol 39:151–157
16. Berrichi A, Yalaoui F, Amodeo L, Mezghiche M (2010) Bi-objective ant colony optimization approach to optimize production and maintenance scheduling. Comput Oper Res 37 (9):1584–1596
17. Graham RL, Lawler EL, Lenstra JK, Kan AHGR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326
18. Rad SF, Ruiz R, Boroojerdian N (2009) New high performing heuristics for minimizing makespan in permutation flowshops. Omega 37(2):331–345
19. Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Europ J Oper Res 47(1):65–74

20. Ruiz R, Stutzle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Europ J Oper Res 177(3):2033–2049
21. Taillard E (1993) Benchmarks for basic scheduling problems. Europ J Oper Res 64(2):278–285
22. Ruiz R, Maroto C, Alcaraz J (2006) Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34 (5):461–476
23. Zobolas GI, Tarantilis CD, Ioannou G (2009) Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Comput Oper Res 36(4):1249–1267
24. Peace GS (1993) Taugchi methods: a hands-on approach. Addison-Wesley, Reading, MA
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(2):426–438