Ant Colony Optimization for the Single Machine Total Earliness Tardiness Scheduling Problem

Rym M'Hallah[1] and Ali Alhajraf[2]

[1] Department of Statistics and Operations Research, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait
mhallah@kuc01.kuniv.edu.kw
[2] af_alhajraf@kuc01.kuniv.edu.kw

Abstract. This paper proposes an ant colony optimization hybrid heuristic (ACH) for the total earliness tardiness single machine scheduling problem where jobs have different processing times and distinct due dates, and the machine can not be idle. ACH is an ant colony system with daemon actions that intensify the search around good quality solutions. The computational results show the effectiveness of ACH.

Keywords: *ant colonies, earliness, tardiness, scheduling, heuristics.*

1 Introduction

The globalization of the world's economy along with the internet's development have induced many changes in today's industries. These changes have shifted the industry from mass to just-in-time production. Subsequently, competitors are now forced to satisfy their customers' demands as close as possible to their due-dates. Missing a demand's due-date may result in the loss of the customer or in penalties whereas satisfying a client's demand earlier than its due-date may cause unwanted inventory or product deterioration. Thus, companies need to minimize the total earliness tardiness (TET) of scheduled jobs.

The NP hard single machine TET scheduling problem, $1|d_j|\sum E_j + T_j$, consists in searching for an optimal sequence of a set $N = \{1, \ldots, n\}$ of independent jobs to be scheduled on a single machine. Each job is characterized by its processing time p_j and due date d_j. All jobs are ready for processing at time zero and job preemption is not allowed. Furthermore, no idle time is allowed on the machine whose capacity is limited compared to demand. For a particular permutation of N, the completion time C_j of job $j, j \in N$, is the sum of the processing times of the subset of jobs that precede j including j. Job j is early if $C_j \leq d_j$ with earliness $E_j = max\{0, d_j - C_j\}$, but is tardy if $C_j > d_j$ with tardiness $T_j = max\{0, C_j - d_j\}$. To be on time, jobs with close due dates have to compete for the same time slot [3]. Thus, only small sized instances of the mixed integer linear model of the problem can be solved exactly [14].

The $1|d_j|\sum E_j + T_j$ problem has been tackled using exact and approximate algorithms. Exact methods are based on dynamic programming [1] and branch

N.T. Nguyen et al. (Eds.): IEA/AIE 2008, LNAI 5027, pp. 397–407, 2008.

and bound [12] whereas approximate ones are based on dispatching rules and metaheuristics. Ow and Morton [15] present a filtered beam search –with no backtracking– that uses a linear and an exponential priority rule to guide the search. Almeida and Centeno [3] combine tabu search, simulated annealing, and hill climbing to generate near optima. Ventura and Radhakrishnan [20] apply Lagrangean relaxation and subgradient optimization to their binary formulation. Valente and Alves [18] design a dispatching rule and a greedy algorithm based on a lookahead parameter. Valente and Alves [19] compare their filtered and recovering beam search to existing heuristics. Finally, M'Hallah [14] proposes a hybrid heuristic (HH) that combines local (dispatching rules, hill climbing and simulated annealing) and global (genetic algorithms) search.

This paper tackles the $1|d_j|\sum E_j + T_j$ using a heuristic inspired from ant colony optimization (ACO). ACO, which mimics the foraging behavior of real ants [2], converges to optimality under certain conditions [8]. It has been successfully applied to a large variety of combinatorial optimization problems [4] including single machine [6], flow shop [21], and job shop [10] scheduling. Its widespread application is due to its success in addressing the two competing goals of metaheuristics: exploration and exploitation. Exploration or evolution allows diversification of the solution space whereas exploitation or learning preserves the good parts of near optima and intensifies the search around them.

Section 2 explains the basic mechanisms of ant colony algorithms. Section 3 details the proposed heuristic. Section 4 tunes its parameters and evaluates its performance. Finally, section 5 is a summary.

2 Ant Colony Algorithms

ACO mimics the behavior of real ants, which are known for their complex social behavior and cooperative work despite their blindness [5]. Ants identify the shortest path between their nest and a food source without using any visual cue [7]. They exchange information regarding a food source by laying, on their path between their nest and a food source, an odorous chemical substance, pheromone. Different ants searching for food at a later time sense the pheromone left by earlier ants and tend to follow a trail with a strong pheromone concentration in hope that it leads them to a food source fast. They choose their path by a probabilistic decision biased by the amount of pheromone: the larger the amount of pheromone on the trail, the higher the probability that ants follow it. In the absence of a pheromone trail, ants move randomly. As time evolves, shorter paths between the nest and a food source have a high traffic density whereas pheromone evaporates along low-density paths. This behavior leads to a self-reinforcing process, which in turn leads to the identification of the shortest path between the nest and a food source [2].

The most frequently applied ACO algorithms are the ant system (AS) and the ant colony system (ACS). When applied to the traveling salesman problem [8], AS positions a set $\mathbf{A}$ of ants randomly on the network. An ant $h \in \mathbf{A}$ moves from a node i to a node j according to a probability p_{ij}^h, which depends on η_{ij}, the

arc's attractiveness according to its contribution toward the objective function, and on π_{ij}, the learned desirability of moving from i to j :

$$p_{ij}^h = \begin{cases} \pi_{ij}^\alpha \eta_{ij}^\beta / (\sum_{j' \in N_i^h} \pi_{ij'}^\alpha \eta_{ij'}^\beta) & \text{if } j \in N_i^h \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where N_i^h is the feasible set of destinations of h when located at node i, and α and β are two parameters that weigh the relative importance of π_{ij} and η_{ij}. Once all ants have completed their tours, the length of each tour is evaluated, and the pheromone amounts on the network are updated using evaporation and deposit. Evaporation reduces all pheromone amounts on the network by a constant evaporation rate $\varphi \in (0, 1]$:

$$\pi_{ij} = (1 - \varphi)\pi_{ij}. \tag{2}$$

Pheromone deposit increases the pheromone level of each traveled network arc (i, j) :

$$\pi_{ij} = \pi_{ij} + \sum_{h \in \mathbf{A}} \Delta\pi_{ij}^h. \tag{3}$$

That is, every time an ant $h \in \mathbf{A}$ travels (i, j), π_{ij} increases by $\Delta\pi_{ij}^h$, which is a function of z_h, the solution value of h. Subsequently, AS repositions the ants randomly in the system where ant $h \in \mathbf{A}$ moves from i to j according to the updated p_{ij}^h. The process is repeated until the stopping criterion is met.

AS does not use any centralized daemon actions; i.e., actions not performed by the ants but by an external agent [7]. For real ants, daemon actions can be the wind moving a food source closer or further, floods forbidding some paths, etc. [7]. In artificial systems, daemon actions can be simple such as depositing additional pheromone along the path followed by the incumbent, or more elaborate such as local search procedures to improve the population's fitness, to prohibit some paths, or to reinforce a set of complex constraints [6,7,17].

ACS is a more sophisticated version of AS combining exploration and exploitation [5,16] . Ant $h \in \mathbf{A}$ moves from i to j according to p_{ij}^h, which reflects a trade-off between the exploration of new connections and the exploitation of available information [7]. ACS generates a random number q and compares it to q_0, an ACS parameter. If $q \leq q_0$, then ACS chooses j such that

$$\pi_{ij}^\alpha \eta_{ij}^\beta = \max_{j' \in N_i^h} \{\pi_{ij'}^\alpha \eta_{ij'}^\beta\} \tag{4}$$

i.e., ACS exploits the available knowledge, choosing the best option with respect to the weighted heuristic and pheromone information. Otherwise, ACS applies a controlled exploration as in AS; i.e., it computes p_{ij}^h using (1). Once all ants have completed their tours, ACS updates the pheromone levels using (2) and (3). In addition, to diversify the solution space and avoid premature convergence, ACS applies an online step-by-step pheromone trail update (including

both pheromone evaporation and deposit) [7]. Each time ant h moves from i to j, ACS decreases the pheromone amount on the arc (i, j) making it less attractive to the following ants:

$$\pi_{ij} = (1 - \varphi)\pi_{ij} + \varphi\pi_0,$$

where π_0 is a lower bound on the pheromone amount along (i, j). Finally, to exploit the available information, ACS applies a daemon action which updates the pheromone level along the path traveled by the ant corresponding to the current incumbent solution.

Different variations of AS and ACS have been applied to scheduling problems. Liao and Juan [11] introduce a new initial pheromone trail parameter to ACO and apply it to the single machine weighted tardiness scheduling problem with sequence-dependent setup times. Tasgetiren et al. [17] test a swarm optimization algorithm, that uses a smallest position value and variable neighborhood search (VNS) as daemon actions, on the permutation flow shop problem with the objective of minimizing both the makespan and the total flow time. Gutjahra and Rauner [9] apply ACO to a dynamic regional nurse scheduling problem where the daily assignment of pool nurses to public hospitals takes into account many soft and hard constraints (eg., shift dates and times, working patterns, nurses qualifications, nurses and hospitals preferences, and costs). Ross and Dini [16] propose an ACS for a flexible manufacturing system in a job-shop environment with routing flexibility, sequence-dependent setup and transportation time. Lo et al. [13] present a modified AS for the precedence and resource-constrained multiprocessor scheduling problems where AS solves the scheduling problems while a dynamic heuristic assigns jobs to processors, and satisfies the time-dependency structure. Herein, an ant colony heuristic (ACH) is proposed for the $1|d_j|\sum E_j + T_j$.

3 The Proposed Heuristic

ACH assimilates the move of an ant from i to j to assigning job j to position i. An ant stops when it schedules all n jobs. Initially, the system constructs $\mathbf{A}$ by choosing $m = |\mathbf{A}|$ random sequences of the n jobs, and assigning each of them to an ant. It evaluates z_h, the TET of ant h, $h \in \mathbf{A}$, and identifies the best current solution h^* whose value $\overline{z} = \min_{h=1,m} \{z_h\}$.

ACH quantifies its acquired knowledge about the problem by setting π_{ij} equal to the proportion of times job j appears in position i in the best $m/5$ ants of the current generation. It then builds m new artificial ants. It sets ant h, $h \in \mathbf{A}$, equal to the empty sequence, and $N_1^h = N$, where N_i^h is the set of candidate jobs for position i for ant h, which has the ordered set of jobs $\overline{N}_{i-1}^h$ assigned in positions $1, \ldots, i-1$. That is, $N_i^h \bigcup \overline{N}_{i-1}^h = N$.

For ant h, the assignment of job j to position i depends on the acquired knowledge π_{ij} and the assignment's attractiveness

$$\eta_{ij} = 1 - (E_j + T_j) / \max_{j' \in N_i^h} \{E_{j'} + T_{j'}\}. \tag{5}$$

ACH draws a random number q from the continuous Uniform[0,1], and compares it to q_0, a threshold level for intensification. If $q \leq q_0$, ACH chooses j according to (4); otherwise, it opts for the best local option choosing the job j with the largest opportunity; that is, $\eta_{ij} = \max_{j' \in N_i^h} \{\eta_{ij'}\}$. It defines the set of jobs that remain to be positioned setting $\overline{N}_i^h = \overline{N}_{i-1}^h \setminus \{j\}$, and the set of positioned jobs as $N_{i+1}^h = N_i^h \bigcup \{j\}$. It continues this process till all jobs have been assigned.

Subsequently, ACH updates the learned desirability π_{ij} using pheromone evaporation and deposit. If job j is assigned to position i, then ACH sets

$$\pi_{ij} = (1 - \varphi)\pi_{ij} + \varphi(1 - (E_j + T_j)/z_h). \tag{6}$$

The amount of pheromone deposited is proportional to the impact of assigning j to i on z_h. The smaller $(E_j + T_j)$ relative to z_h, the higher is the amount of pheromone deposited.

Once the m ants are obtained, the populations of the current and the previous generations are merged, and the best m ants are retained for further investigation. Indeed, each of the best m ants is subject to an intensified search. Ant h, $h \in \mathbf{A}$, is subject to $(n-1)$ two-opt swaps. If any of the $n-1$ neighbors improves z_h, then it replaces h in the current generation. This intensification step, or daemon action, is needed to speed ACH's convergence.

Finally, ACH applies a global pheromone update. For every (i, j), it evaporates the pheromone using (2), and deposits Δ_{ij}, where Δ_{ij} is the proportion of times j has been assigned to i in the best $m/5$ ants of the current generation. Subsequently, ACH updates h^* and $\overline{z}$, and repeats these steps for a prefixed number of generations, n_g. A summary of ACH is given in Algorithm 1.

4 Computational Results

The objective of the computational experimentation is twofold: (i) to tune ACH's parameters and investigate their impact on ACH's performance, and (ii) to compare ACH's performance to a standard ACO algorithm (AS) and to the solutions obtained by Cplex. ACH and AS are coded using Fortran, under the Microsoft Developer Studio platform whereas Cplex is evoked from GAMS. All computation is undertaken on a Pentium IV 3.0 GHz and 512 MB of RAM.

To tune ACH's parameters, we set $n = 10, 20, 30$; $n_g = 10, 50, 100, 300$; $m = 100, 500, 1000, 2000, 5000, 10000$; $q_0 = .1, .3, .5, .7, .9$; $\rho = .1, .3, .5, .7, .9$; and $(\alpha, \beta) = (1, 1), (1, 2), (2, 1)$. For each possible combination of these parameters and level n, we generate ten instances as in [15]; that is, the processing times and due dates are random integers from the Uniform $[1, 12]$, and $[.65\overline{p}, 1.15\overline{p}]$, respectively, where $\overline{p} = \sum_{j \in N} p_j$. We run each instance ten times using ACH, and compute RT, the average run time (in seconds) over the ten replications, and the performance ratio $r = \overline{z}/z^*$, where $\overline{z}$ is ACH's average solution value over the ten replications, and z^* is the exact solution value obtained via Cplex.

Algorithm 1. Detailed algorithm of ACH

Initialization

1. For $h = 1, \ldots, m$
   (a) Set ant h to a random permutation of the n jobs.
   (b) Compute z_h, the total earliness tardiness of h.
2. Set $\overline{z} = \min_{h=1,m} \{z_h\}$, and h^* the ant whose $z_{h^*} = \overline{z}$.
3. Set $g = 1$.
4. Set π_{ij}, the learned desirability of assigning job j to position i, to the proportion of times job j appears in position i in the best $m/5$ ants of the current generation.

Iterative Step

1. For $h = 1, \ldots, m$
   (a) Set $N_1^h = N$, $\overline{N}_0^h = \emptyset$.
   (b) For $i = 1, \ldots, n$
      – Generate a random number q from the continuous Uniform[0,1].
      – If $q \leq q_0$, choose j according to (4); else, choose $j \ni \eta_{ij} = \max_{j' \in N_i^h} \{\eta_{ij'}\}$.
      – Set $\overline{N}_i^h = \overline{N}_{i-1}^h \setminus \{j\}$, and $N_{i+1}^h = N_i^h \bigcup \{j\}$.
   (c) Update π_{ij} using (6).
2. Choose the best m ants out of the m ants of the current generation and the m ants of the previous generation.
3. For $h = 1, \ldots, m$
   (a) Generate $(n-1)$ neighbors of ant h using two-opt swaps;
   (b) Choose the best out of the n solutions to replace h in the current generation.
4. Determine Δ_{ij}, the proportion of times j has been assigned to i in the best $m/5$ ants of the current generation.
5. Set $\pi_{ij} = (1 - \varphi)\pi_{ij} + \Delta_{ij}$.
6. Update h^* and $\overline{z}$.
7. Set $g = g + 1$.

Stopping condition

If $g > n_g$, stop; otherwise, goto the **iterative step**.

Figures 1 and 2 show that increasing n_g and m improves r; thus improves ACH's opportunity to converge toward a global optimum; however, this occurs at the cost of a larger runtime. They further indicate that setting $n_g = 300$ and $m = 5000$ seems a reasonable tradeoff between solution quality and runtime. ACH is using a relatively large number of ants; however opting for $m = 10$ with $n_g = 300$ yields an average r equal to 1.065 versus 1.016 when $m = 5000$. This is most likely due to the competition of the jobs for the same time slots on the machine, and the resulting myopic decisions of the ants.

Figure 3 shows that there is no clear rule of thumb to privileging either the accumulated knowledge or the local heuristic information for different population sizes. For example, $\alpha = \beta = 1$ is a better alternative when $4000 \leq m \leq 6000$, but is not the best strategy for lower or higher values of m. It is therefore recommended that preliminary testing be undertaken prior to setting α and β.

Figure 4 displays the effect of q_0 and φ on ACH's performance. Even though no clear rule of thumb applies, some general guidelines can be established. A high φ may cause too much pheromone evaporation along good paths and too much pheromone deposit along undesirable ones; thus, good quality knowledge may be lost while myopic decisions are strengthened. Indeed, for TET, a "good" position for a job strongly depends on the jobs assigned to the preceding positions. On the

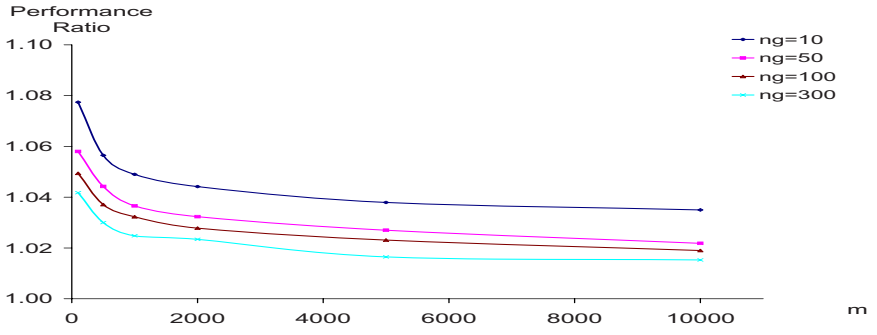

Fig. 1. ACH's solution quality as n_g and m vary ($n = 20, \alpha = \beta = 1, \varphi = 0.9, q_0 = .1$)

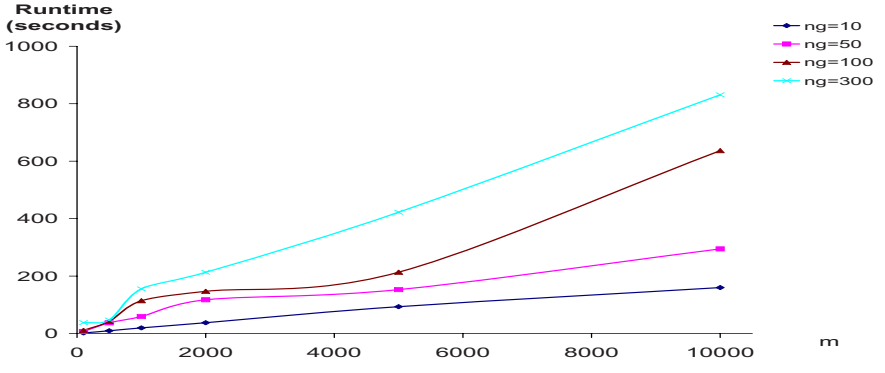

Fig. 2. ACH's Runtime as n_g and m vary ($n = 20, \alpha = \beta = 1, \phi = 0.9, q_0 = .1$)

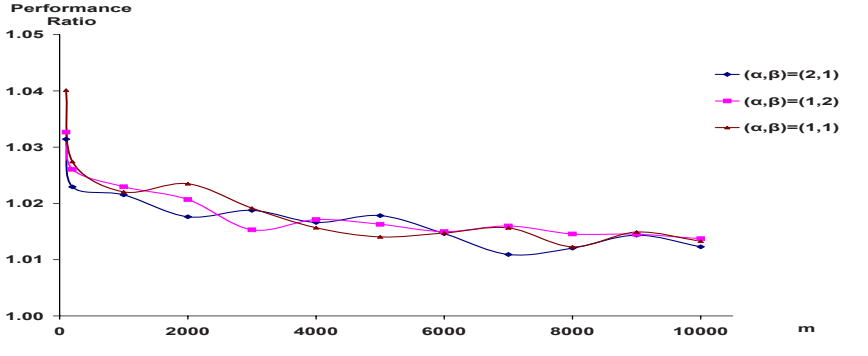

Fig. 3. ACH's Performance as (α, β) varies ($n = 20, n_g = 300, q_0 = 0.9, \varphi = 0.9$)

other hand, a low φ limits the acquisition of new knowledge and relies heavily on available information; thus, may cause premature convergence and stagnation

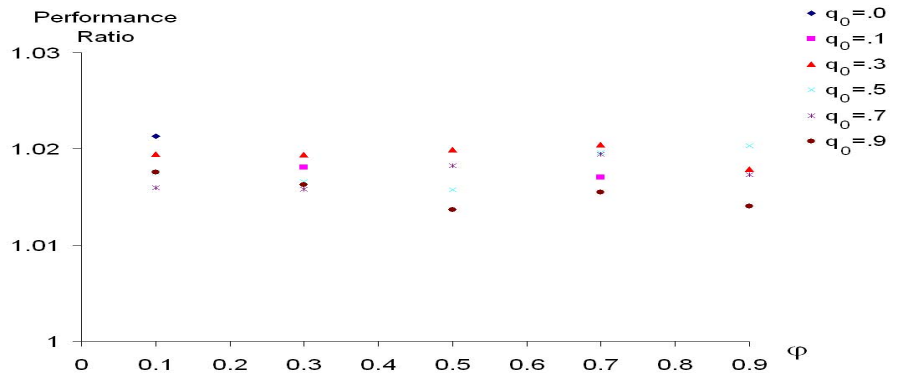

Fig. 4. ACH's Performance as q_0 and φ vary ($n = 20, \alpha = \beta = 1, n_g = 300, m = 5000$)

of the algorithm. $\varphi = 0.5$ seems to offer a good balance between pheromone evaporation and deposit; i.e., between maintaining prior knowledge and using newly acquired information (obtained via heuristic information).

A high q_0 enhances the use of available knowledge, and limits relying on local search. It guides ACH toward the best decision according to a weighted version of the accumulated (pheromone) and acquired (heuristic) knowledge. On the other hand, a very low q_0 encourages a guided exploration of the search space where the search is biased towards more promising areas (guided by prior knowledge). Despite few exceptions, setting $q_0 = 0.9$, and $\varphi = .5$ seems to yields good results.

ACH's runtime, whose evolution as a function of n is illustrated in Figure 5, can be approximated by $RT = .2127n^{2.2314}$ when $m = 5000$ and $n_g = 300$. Initially, ACH creates, evaluates, and sorts m ants. In addition, in each of the n_g generations, ACH undertakes the following steps. It creates another m ants, where each ant requires n comparisons. It then sorts $2m$ ants in $O(2mlog(2m))$. Finally, it applies a two-opt neighborhood search for each of the best m ants, where each search requires $(n-1)nm$ operations. Subsequently, the number of operations undertaken by ACH is $O(2n_g(n^2 + m(n + log(2m))))$.

The performance ratios for $n = 10, 20$, and 30 are on average around 1.01 when $m = 5000, n_g = 300, \varphi = 0.5,\ q_0 = 0.9$, and $\alpha = \beta = 1$, with the average r equaling the median. The average r is 1.00 for $n = 10$, 1.01 for $n = 20$ and 1.02 for $n = 30$. This is expected since m and n_g are fixed; thus, the smaller the problem size, the higher the chances of ACH to find the global optimum.

ACH's convergence rate may be improved as follows. The initial sequences being randomly generated in Step 1(a) of the initialization step of ACH may be substituted by their best neighbors obtained via simulated annealing or tabu search. In addition, in Step 3(a) of the iterative step, the $n-1$ neighbors can be obtained by applying a simulated annealing $n-1$ times to the current ant h.

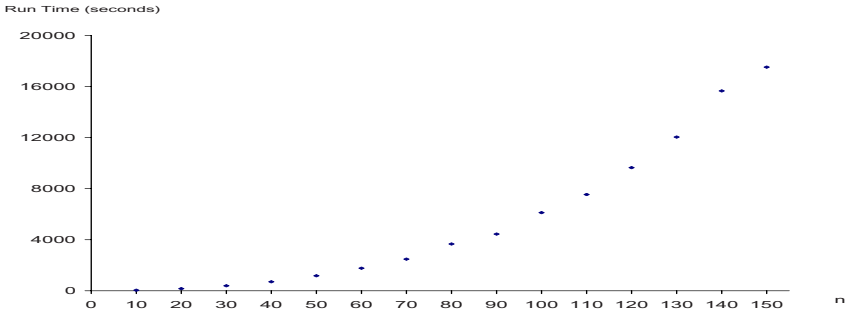

Fig. 5. ACH's Run Time as n increases ($\alpha = \beta = 1, n_g = 300, m = 5000, q_0 = .9, \varphi = .1$)

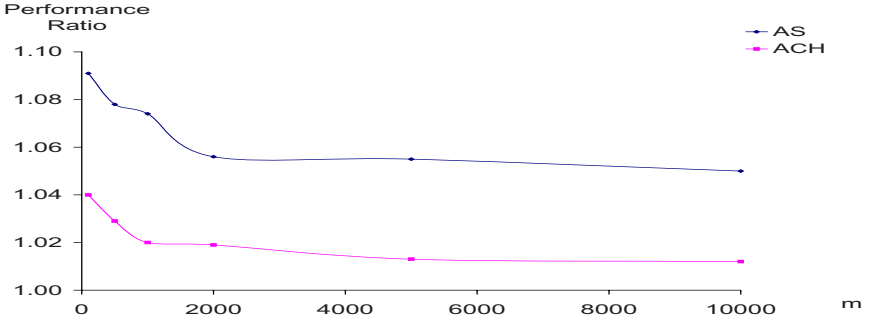

Fig. 6. Comparison of ACH and AS's performance as m increases ($\alpha = \beta = 1, n_g = 300, q_0 = .9, \varphi = .1, n = 20$)

These modifications will intensify the search around promising areas and improve the quality of the knowledge being propagated through ACH's generations.

The comparison of ACH to AS shows that hybridization improved ACH's performance. The average improvement is 4.5% and reaches 5.4% when $m = 1000$; yet, this improvement is independent of the problem size. Figure 6 further shows that ACH yields consistently better results than AS.

5 Conclusion

This paper proposes an ant colony optimization hybrid heuristic for the minimum total earliness tardiness single machine scheduling problem with distinct deterministic due dates. The heuristic is a modified ant colony system with daemon actions that intensify the search in promising areas. The computational results illustrate the heuristic's good performance. The heuristic can be further enhanced if implemented as a parallel algorithm.

References

1. Abdul-Razaq, T.S., Potts, C.N.: Dynamic programming state-space relaxation for single machine scheduling. Journal of the Operational Research Society 39, 141–152 (1988)
2. Albritton, M.D., McMullen, P.R.: Optimal product design using a colony of virtual ants. European Journal of Operational Research 176, 498–520 (2007)
3. Almeida, M.T., Centeno, M.: A composite heuristic for the single machine early tardy job scheduling problem. Computers & Operations Research 25, 535–625 (1998)
4. Bullnheimer, B., Hartl, R.F., Strauss, C.: An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 89, 319–328 (1999)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
6. Gagné, C., Price, W.L., Gravel, M.: Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society 53, 895–906 (2002)
7. García-Martínez, C., Cordón, O., Herrera, F.: A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. European Journal of Operational Research 180, 116–148 (2007)
8. Gutjahr, W.J.: A graph-based ant system and its convergence. Future Generation Computer Systems 16, 873–888 (2000)
9. Gutjahr, W.J., Rauner, M.S.: An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria. Computers & Operations Research 34, 642–666 (2007)
10. Heinonen, J., Pettersson, F.: Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. Applied Mathematics and Computation 187(2), 989–998 (2007)
11. Liao, C.J., Juan, H.C.: An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research 34, 1899–1909 (2007)
12. Liaw, C.F.: A branch and bound algorithm for the single machine earliness and tardiness scheduling problem. Computers & Operations Research 26, 679–693 (1999)
13. Lo, S.T., Chen, R.M., Huang, Y.M., Wu, C.L.: Multiprocessor system scheduling with precedence and resource constraints using an enhanced ant colony system. Expert Systems with Applications 34(3), 2071–2081 (2008)
14. M'Hallah, R.: Minimizing Total Earliness and Tardiness on a Single Machine Using a Hybrid Heuristic. Computers & Operations Research 34(10), 3126–3142 (2007)
15. Ow, P.S., Morton, E.T.: The single machine early/tardy problem. Management Science 35, 177–191 (1989)
16. Rossi, A., Dini, G.: Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. Robotics and Computer-Integrated Manufacturing 23, 503–516 (2007)
17. Tasgetiren, M.F., Liang, Y., Sevkli, M., Gencyilmaz, G.: A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research 177, 1930–1947 (2007)
18. Valente, J.M.S., Alves, R.A.F.S.: Filtered and Recovering beam search algorithms for the early/tardy scheduling problem with no idle time. Computers & Industrial Engineering 48(2), 363–375 (2005)

19. Valente, J.M.S., Alves, R.A.F.S.: Improved heuristics for the early/tardy scheduling problem with no idle time. Computers & Operations Research 32(3), 557–569 (2005)
20. Ventura, J.A., Radhakrishnan, S.: Single machine scheduling with symetric earliness and tardiness penalties. European Journal of Operational Research 144, 598–612 (2003)
21. Ying, G.C., Liao, C.J.: Ant colony system for permutation flow-shop sequencing. Computers & Operations Research 31, 791–801 (2004)