Int. J. Production Economics 120 (2009) 287–300

Contents lists available at ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

Ant colony optimization for the single model U-type assembly line balancing problem

Ihsan Sabuncuoglu [a,*], Erdal Erel [b,1], Arda Alp [c,2,3]

[a] Department of Industrial Engineering, Bilkent University, Bilkent 06800, Ankara, Turkey
[b] Faculty of Business Administration, Bilkent University, Bilkent 06800, Ankara, Turkey
[c] Artificial Intelligence Laboratory, School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 10 April 2007
Accepted 14 November 2008
Available online 19 January 2009

Keywords:
General assignments problem
U-type assembly line balancing problem
Ant colony optimization
Metaheuristic

ABSTRACT

An assembly line is a production line in which units move continuously through a sequence of stations. The assembly line balancing problem is defined as the allocation of tasks to an ordered sequence of stations subject to precedence constraints with the objective of optimizing a performance measure. In this paper, we propose ant colony algorithms to solve the single-model U-type assembly line balancing problem. We conduct an extensive experimental study in which the performance of the proposed algorithm is compared against best known algorithms reported in the literature. The results indicate that the proposed algorithms display very competitive performance against them.

1. Introduction

An assembly line is a production line in which units move continuously through a sequence of stations at which assembly operations (tasks) are performed. The design of an assembly line requires assigning the tasks into stations considering the precedence relations among these tasks that have to be completed within the time interval between two successive outputs. This time interval is called as the cycle time. Such an assignment is accomplished to optimize some line efficiency measure. This is known as the simple assembly line balancing problem (SALBP). In a straight-line configuration, stations are located along a straight-line, whereas in a U-type assembly line, the line is configured into a U-shape topology. The latter problem is defined as the U-type assembly line balancing problem (UALBP).

In a straight line configuration, if task j is performed at station k, then all its direct or indirect predecessors have to be assigned to one of these stations 1,…,k. A task and its indirect predecessors (or successors) can be allocated to the same station if and only if all intermediary tasks are also in the same station. For example, in the precedence diagram given in Fig. 1, tasks 3 and 8 can only be allocated to the same station if and only if tasks 5 and 6 are also allocated to that station. In UALBP, each task and any of its predecessors and/or successors can share the same station but it must be satisfied that, all predecessors and/or successors of task j, performed at station k, have to be assigned to one of the stations 1,…,k. (Miltenburg and Wijngaard, 1994). In these systems, tasks are to be allocated into stations by moving forward and backward through the precedence diagram in contrast to a typical forward move in the traditional assembly systems (Erel et al., 2001). Hence, tasks from the beginning and end of the precedence diagram may be assigned to the same station. In Fig. 2, a straight and a U-line configuration are displayed for the example in Fig. 1. Note that the first and

* Corresponding author. Tel.: +90 312 290 1607; fax: +90 312 266 4054.
E-mail addresses: sabun@bilkent.edu.tr (I. Sabuncuoglu), erel@bilkent.edu.tr (E. Erel), arda.alp@epfl.ch (A. Alp).

[1] Tel.: +90 312 290 1597.

[2] Tel.: +41 21 693 6595.

[3] This research was done when Arda Alp was a student at Bilkent University.

0925-5273/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2008.11.017

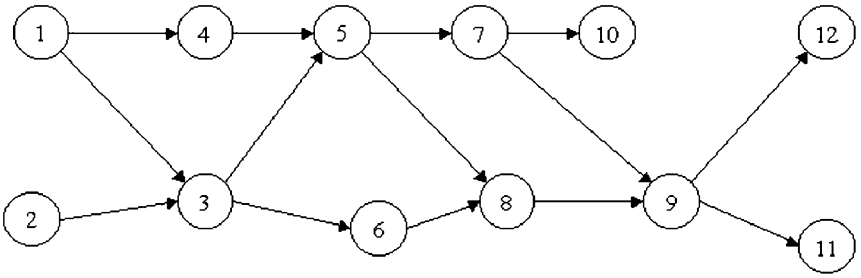

Fig. 1. Example of a precedence diagram (Scholl and Klein, 1999).

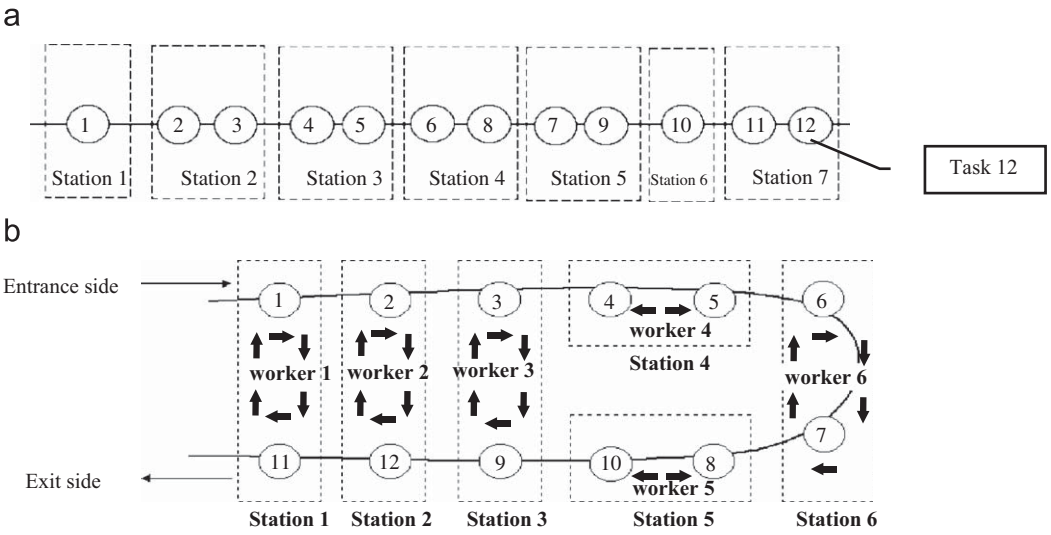

Fig. 2. Straight and U-line configurations for the example in Fig. 1. (a) Straight line configuration and (b) U-line configuration.

the last tasks in the precedence diagram are assigned to the same stations.

In U-shape allocations, under unstable conditions, it is more effective and easier to reallocate the workers to balance the work load. For the cases in which demand fluctuations are high (competitive markets), the flexibility built in U-lines provides improvement in the performance measures and adaptability to changing conditions. Thus, U-lines are typically preferred over traditional straight lines (Miltenburg and Wijngaard, 1994). On the other hand, the U-line balancing problem is more challenging than its counterpart, due to the larger solution space and comparably more possibilities provided by the U-shape allocation.

In this study, we modify ant colony algorithms to solve the U-type assembly line balancing problem. The problem considered in this study is a single model, deterministic U-line balancing problem. Our objective is to find a design with the minimum number of stations subject to the cycle time and precedence relations constraints. This problem has been extensively studied in the literature and several solution approaches have been proposed including simulated annealing (SA), genetic algorithms (GA), branch and bound based algorithms, etc. The purpose of this study is to see the feasibility and effectiveness of ant colony approach which is one of the most recent meta-heuristics to solve this well known problem and compare it against others in terms of quality of solution and solution time.

The rest of the paper is organized as follows. Relevant literature is given in Section 2. This is followed by the structure of the proposed algorithm in Section 3. The experimental setting is given in Section 4. Computational results are discussed in Section 5. Finally, concluding remarks and future research directions are given in Section 6.

2. Literature review

This section is divided into two parts: (1) the basic studies on ant colony optimization (ACO) and (2) the relevant studies on the U-line balancing problem.

2.1. ACO heuristic

The first ant algorithm, *Ant System* (AS), is proposed by Colorni et al. (1991, 1992) and used to solve the well-known travelling salesman problem (TSP). Although the algorithm developed by Dorigo et al. (1996) cannot compete with the state-of-the-art algorithms for large size TSP instances, it stimulated further research in this area.

Gambardella and Dorigo (1995), Dorigo and Gambardella (1996) analyse some properties of AS and its sensitivity to various parameters for the TSP. After the development of Elitist Ant System (AS_{elite}), which is an extended version of AS, Gambardella and Dorigo (1995) proposed Ant Colony System (ACS) as an extension of AS for the TSP. The computational studies by Dorigo and Gambardella (1997a, b) indicate that ACS outperforms other nature-inspired algorithms such as SA (Johnson et al., 1989, 1991) and GA (Holland, 1975).

In another study, Stützle (1998) developed a fast local search procedure, MAX–MIN ant system, derived from AS, for the flow shop problem. In a different study, Stützle and Dorigo (1999) provided further information about the MAX–MIN ant system.

Bullnheimer et al. (1999) introduced a new rank-based version of the AS (AS_{rank}) for the TSP. Their results indicate that AS compete well with two other metaheuristics, SA and GA, and it outperforms the other methods (AS, AS_{elite}) for large problems in terms of average and especially the worst case behavior.

Fenet and Hassas (2000) proposed a procedure that work with mobile reactive agents and introduce a successful, new problem-solving framework, A.N.T., a distributed problem solving with local interactions of ants.

In the following years several researches worked on different features of ant algorithms. Montgomery and Randall (2002) worked on alternative pheromone representations for ACO and Middendorf et al. (2002) proposed multi colony ant algorithms (MCAA). McMullen and Tarasewich (2003) presented a heuristic for the assembly line balancing problem using concepts derived from ACO techniques.

Several other ACO heuristics are also proposed for different problems including: single machine total weighted tardiness problem (Besten et al., 2000a, b); quadratic assignment and frequency assignment problem (Maniezzo and Carbonaro, 2001; Middendorf et al., 2002); an industrial scheduling problem (Gagné et al., 2001); first order parallel shop scheduling problem (Blum and Sampels, 2002); flowshop scheduling problem (T'kindt et al., 2002; Stützle, 1998; Lin et al., 2008; Gajpal and Rajendran, 2006); assembly line balancing problem (Bautista and Pereira, 2002, 2007; McMullen and Tarasewich, 2003). All these studies demonstrate how ant algorithms can be used to solve combinatorial optimization problems. The reader can refer to the study of Krishnaiyer and Cheraghi (2002) for applications of ant algorithms. A general overview of ACO can be found in Dorigo and Stützle (2004), Dorigo and Socha (2007), and Dorigo and Blum (2005).

2.1.1. ACO to solve assembly line balancing problems

Bautista and Pereira (2002) propose an ACO heuristic to solve the SALBP based on the AS heuristic of Dorigo et al. (1991a, 1996). The proposed heuristic has a number of versions based on the different policies (e.g., trail information usage policies, trail accumulation policies, use of local search policies). Experiments conducted on the 267-instance problem set of Scholl (1993) show that 182 instances are solved optimally; the mean variation between the optimal solution and the obtained solution ranges from 0.379% to 0.475% among the different versions of the heuristic.

Bautista and Pereira (2007) later develop another ant algorithm again based on the AS heuristic of Dorigo et al. (1996) to solve the time and space constrained assembly line balancing problem (TSALBP). This version of the problem considers spatial constraints around the lines; the motivation stems from automotive assembly lines in which several models are assembled simultaneously. Their computational experiments indicate that solving TSALBP is relatively more difficult than SALBP. In both of their papers, Bautista and Pereira (2002, 2007) solve a real-life problem taken from the bike assembly and automotive industry, respectively.

McMullen and Tarasewich (2003) developed four heuristic procedures based on ACO techniques to solve the mixed-model, stochastic assembly line balancing problems with task paralleling. They compare the procedure against other heuristics for the objectives of probability of jobs being completed on time, cycle time ratio (ratio of the desired cycle time to the actual cycle time), and cost of workers and machines. The experiments show mixed results depending on the objectives considered and the problem parameters.

2.2. U-type line balancing

U-line is a relatively new and promising topic in the assembly line balancing literature. The first study is due to Miltenburg and Wijngaard (1994) who proposed a dynamic programming formulation to solve 21 relatively small problems (with up to 11 tasks). The authors also develop a heuristic procedure based on the maximum ranked positional weight (RPWT) for large size problems (with up to 111 tasks). Later, Miltenburg and Sparling (1995) developed three exact algorithms for the UALBP: a reaching dynamic programming algorithm, breadth- and depth-first branch-and-bound algorithms. The authors solve 180 problem instances with up to 40 tasks. In another study, Urban (1998) developed an integer programming formulation for UALBP and solve problems with up to 45 tasks.

To handle larger problems, Scholl and Klein (1999) propose ULINO (U-line optimizer); a new branch-and-bound procedure that performs a depth-first search by considering bounds and some dominance rules. Computational experience is presented for 256 instances (complete data set) with up to 297 tasks. The results indicate that the proposed method yields very good results for type 1 problem (minimizing the number of stations given the cycle time) and type 2 problem (minimizing the cycle time given the number of stations) in limited computation time.

Erel et al. (2001) developed an SA-based algorithm for UALBP. The proposed algorithm employs an intelligent mechanism to search the large solution space effectively. Their computational results indicate that the proposed method is quite effective in solving the problem and its

success can be attributed to the intelligent way of searching a larger search space.

Sparling and Miltenburg (1998), Miltenburg (1998), Sparling (1998), Kim et al. (2000), and Kara et al. (2007) examined the mixed-model version of ULBP in which various models are assembled one after another on the same line. There are also studies which investigate various properties of the U-shaped production lines. Nakade and Ohno (1999) analyzed the optimal worker allocation problem for a U-line. Miltenburg (2000) investigated the effect of 'U-shape of the line' on the line's effectiveness considering the breakdowns. Miltenburg (2001) studied a one-piece flow production system on U-lines and examines the related literature. Nakade and Ohno (2003) considered a U-line with multiple workers and two types of allocations of workers: a separate allocation and a carousel allocation. Guerriero and Miltenburg (2003), Urban and Chiang (2006), and Chiang and Urban (2006) examined the stochastic ULBP's in which the constant-task-performance-time assumption is relaxed. Aase et al. (2003) developed a family of branch-and-bound procedures to solve the UALBP and determined the design elements to include in the procedures to enhance their performance. Later, the same authors (Aase et al., 2004) studied the effect of adopting UALB on labor productivity and report that switching to UALB's does not improve labor productivity in all cases.

Recently, Boysen and Fliedner (2008) proposed a general solution procedure for assembly line balancing problems (SALBP and UALBP) using ant colony approach. This is the most relevant study to ours. Their procedure consists of two stages: sequence generation and task assignment. The authors use the ant colony approach to construct feasible task sequences in the first stage. Then the allocation of tasks to stations is carried out by well-known mathematical modeling tools (IP, knapsack, etc.) in the second stage. The proposed procedure is considerably versatile in the sense that various line balancing features can be incorporated into the model to solve various assembly line balancing problems encountered in practice. Moreover, the resulting performance of their procedure is quite comparable to the other effective algorithms in the literature. In contrast, in our study, we focus only on UALBP and develop a family of ant colony algorithms to solve this version. Our solution procedures are entirely based on ant colony approach in the sense that the proposed algorithms make both sequencing and task assignment decisions simultaneously, with no need for any mathematical modeling tool. In many ways, these two studies compliment each other in the sense that one utilizes ant colony approach as a part of a general solution procedure, whereas the other is more specific and is solely based on ant colony approach.

3. Proposed approach

3.1. Overview

Ant algorithms are inspired from the collective behavior of ants on the survival of their colonies. While these ants are searching for food, they deposit on the ground a substance called *pheromone* which helps the others to follow the path leading to the food. This is a simple form of indirect communication and interestingly these ants usually find the shortest path between their nest and the food source.

When more ants collectively follow a trail, this trail becomes more attractive for being followed in the future. As more ants are able to pass through shorter paths compared to longer ones (within the same unit of time), more pheromone is likely to accumulate on shorter paths. Therefore, these paths are selected more frequently than the longer ones. This pheromone accumulation mechanism allocates higher priority to the better solutions and this idea is used to direct the search process in 'ant algorithms' and also in our proposed ant colony algorithm. Our agents are the abstraction of real ants based on their ability to communicate and cooperate using artificial pheromone trails. These agents use local and global trail information to find the best solution and have memory to store their past actions. Unlike the real life in which pheromone update is continuous, it is discrete in a simulated environment. In most of the ant algorithms the agents update the pheromone trails at each solution generation.

In our proposed ant algorithm, agents cooperate with each other to find good solutions for UALBP. These agents try to allocate given tasks to stations considering the precedence relations, U-shape topology, task times and cycle time constraints. UALBP is represented as a graph in which nodes and arcs represent tasks and precedence constraints among the tasks, respectively, and the task times are given as node weights. Arc (i,j) between nodes i and j represents the precedence constraint between task i and task j, implying that task i must be completed for task j to get started.

3.2. Generation of a solution

3.2.1. Fundamental steps

Each agent starts with an empty station; a predefined initial state at which a small amount of pheromone is deposited as a default value for entry in trail accumulation matrix. Based on the given precedence relations and cycle time, assignable and available tasks are determined and a search list is created. Available tasks are the ones with all of their predecessors/successors already assigned to stations, whereas the assignable tasks are the available tasks that can fit into the current station. Starting with the first task, these agents start construction of their own solutions from scratch by allocating tasks to stations one by one, step by step until all tasks are (n) are allocated. This means n iterations for each agent. During the allocation of tasks, if there are available tasks but none of them are assignable, then a new station is opened. Allocation of all tasks is completed in one tour when a full solution is constructed. At the end of one tour, m agents collectively create m solutions.

During the search process, a number of agents move through the neighborhood of the solution space by applying a stochastic local search policy to find alternative

allocations, and hopefully the best one. As they move from one task to another, the memory is updated instantly or later depending on the solution quality. Agents gather valuable information about the problem characteristics and their own performance in order to modify the trail values. Note that this is not a direct communication between two or more agents; on the contrary, it is the common usage of gathered information and naturally the most important feature (cooperation). Trail values sustain the experience gained from the agents and influence the solution quality in the consequent tours. The trail value represents the attractiveness of the assignment of a task to a specific station. The probability of assigning a task to a specific station increases with the number of agents that previously assigned the same task to that station. Thus, during the tours, if a task is assigned to a specific station relatively more, then it is more attractive to assign the same task to the same station in the subsequent tours.

Another component that may have a drastic effect on the solution quality is the greedy force. This is defined as the heuristic value (η_i) that gives a prior knowledge on the attractiveness of a task. The higher the heuristic value, the more desirable the task is. In fact, the heuristic value represents a priori run-time local information provided by a source which is different than trail accumulation mechanism (Dorigo and Stützle, 2000). It is a local information used during the task selection process.

Inspired by the work of Miltenburg and Wijngaard (1994) we have chosen the task priority function to represent the greedy force (this function is also called 'U-line maximum RPWT'). The priority of each task is represented by 'forward' and 'backward' positional weights, ($W^f(k)$, $W^b(k)$), which are defined as the sum of the task time of the task and the task times of all the other tasks that succeed or precede it, respectively. Thus, the tasks with large value are prime candidates for assignment.

Each of the agents has its own constraint *satisfaction list*, a simple write-only memory, which stores the state of each task (assigned or unassigned). During a tour, assigned tasks are marked as assigned and will stay unassignable until the end of that tour. During the calculation of task selection probabilities, constraint satisfaction list is used to determine assignable and available tasks.

The construction of solutions proceeds until a prespecified number of tours is completed. General probability function represents the probability of assignment of task i to station j at iteration t for agent k. Here, T_{ij} (trail accumulation) represents the learned desirability of assigning task i to station j; η_i represents the priority value (also called as *visibility*) of task i; t_i is task time of ith task and $allowed_k$ represents the set of tasks that can be assigned to the current station without violating any precedence constraint and the cycle time constraint for agent k. This search policy is directed by agents' memory, pheromone trail information and problem-specific local information.

3.2.2. Illustrative example

The solution construction for an agent within a tour is illustrated with an example. We consider the example problem in Fig. 3 with a cycle time of 8. The numbers above the nodes represent task times. For agent k, the probability of assignment of task i to station j is defined with a general formula (we shorten our formula for simplicity) as follows:

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[T_{ij}(t)]^{\alpha}[\eta_i]^{\beta}}{\sum_{z\in allowed_k}[T_{zj}(t)]^{\alpha}[\eta_z]^{\beta}} & \text{if } i\in allowed_k\\ 0 & \text{otherwise}\end{cases} \quad (1)$$

Recall from Section 1 that in U-shaped lines a task can be assigned to a station only if all its predecessors (or successors) are already assigned. This reveals the feasibility and priority of forward or backward assignment. If a task is forward (backward) assignable, its heuristic value is the forward (backward) positional weight of that task. For example, in iteration 1 of Fig. 3a, tasks 1 and 5 are forward and backward assignable, respectively. In other words, tasks 1 and 5 can be assigned to the upper and lower parts of station 1, respectively. In iteration 2, task 4 becomes forward assignable, since tasks 2 and 3 do not fit into station 1. Note also that task 5 stays as backward assignable in this iteration. This process continues until all the tasks are assigned to the stations. The parameters α and β allow a user to control the relative importance of pheromone trail versus visibility intensity. These parameters affect the performance of the algorithm due to the tradeoff between the trail intensity and the visibility. If $\alpha=0$, the selection probabilities are proportional to $[\eta_i]^{\beta}$ and the tasks with high positional weight values are more likely to be selected. If $\beta=0$, only the trail information affects the selection probabilities. This tradeoff is reflected by the transition probability. Selection of these parameters and their current values are given in Section 4.

Time counter t tracks the iteration number. The example given in Fig. 3a illustrates a complete tour of an agent. For simplicity, T_{ij}, α, β are taken as 1; ↻ represents forward available and assignable tasks, ↺ represents backward available and assignable tasks; ⊖ represents unavailable and unassigned tasks, and ✕ represents assigned tasks. Final task allocation for this example is given in Fig. 3b.

In order to expose the importance of trail accumulation and its effect on the solutions, we illustrate some of different task allocations of agents in Fig. 4a for arbitrary selected 3 tours. Since this is a random allocation (despite the effect of trail matrix), not all the solutions need to show up at every tour. Note that the solution in Fig. 3b is obtained by agent 1 in tour 1, agents 1, 2, and 4 in tour 2, and agents 2 and 3 in tour 3. Trail accumulation at the end of 15th tour is given in the matrix on the right of Fig. 4b; the initial values are given in the matrix on the left. Total number of agents is taken equal to total number of jobs, though these two quantities can be different. In Fig. 4b it is clear that, even after a small number of tours, some tasks are more likely to be assigned to some stations due to the significantly larger accumulation values on some elements of the matrix. For example, the trail accumulation values, $T_{ij}(15)$, for (1,1), (2,3), (3,2), (4,1), (5,2)th elements are relatively larger implying that the related

a

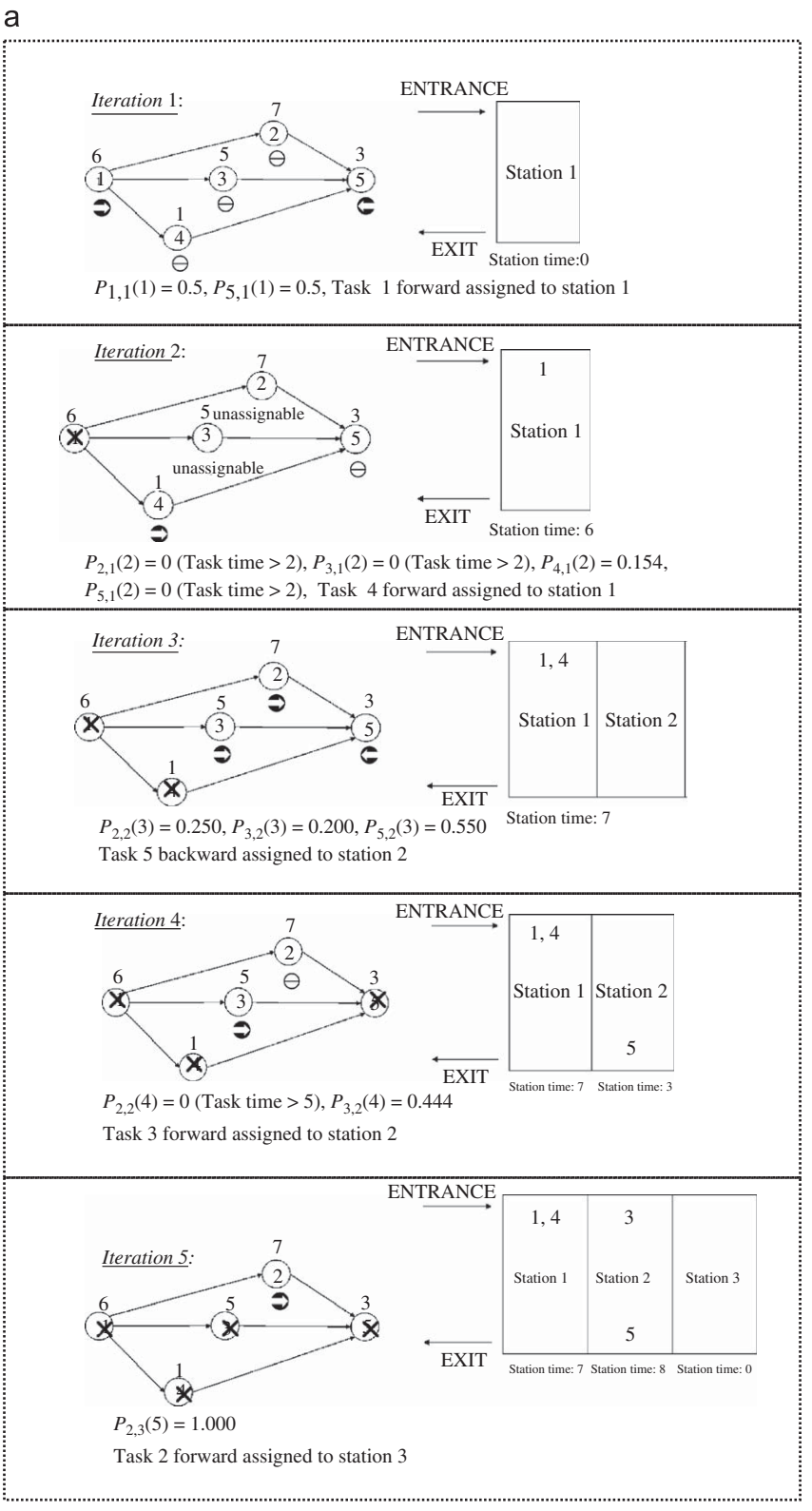

Fig. 3. (a) Task allocations for the example problem and (b) final task allocation for the example problem.

b

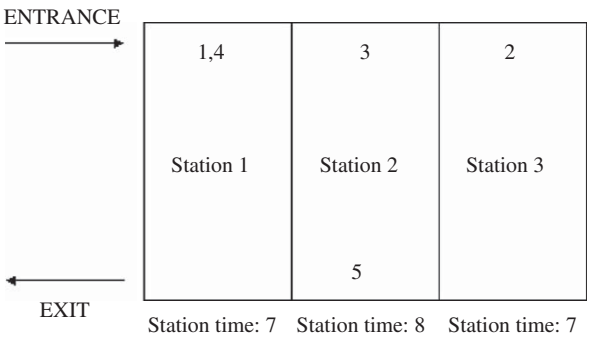

Fig. 3. (*Continued*)

a

Agent 1: ||5, 4 ||3 ||1 ||2 || Station number is 4.
Agent 2: ||5, 4 ||2 ||1 || 3 || Station number is 4.
Agent 3: ||1, 4 ||5, 3 ||2 || - || Station number is 3.
Agent 4: ||1, 4 ||2 ||5, 3|| - || Station number is 3.
Agent 5: ||5, 4 ||2 ||3 || - || Station number is 3.
} Tour 1

Agent 1: ||1, 4 ||5, 3 ||2 || - || Station number is 3.
Agent 2: ||1, 4 ||5, 3 ||2 || - || Station number is 3.
Agent 3: ||5, 4 ||1 ||2 || 3 || Station number is 4.
Agent 4: ||1, 4 ||5, 3 ||2 || - || Station number is 3.
Agent 5: ||5, 3 ||2, 4 ||1 || - || Station number is 3.
} Tour 2

Agent 1: ||5, 3 ||1, 4 ||2 || - || Station number is 3.
Agent 2: ||1, 4 ||5, 3 ||2 || - || Station number is 3.
Agent 3: ||1, 4 ||5, 3 ||2 || - || Station number is 3.
Agent 4: ||1, 4 ||2, 5 ||3 || - || Station number is 3.
Agent 5: ||5, 3 ||2, 4 ||1 || - || Station number is 3.
} Tour 3

b

	Station 1	Station 2	Station 3	Station 4	Station 5
Task 1	1.0000	1.0000	1.0000	1.0000	1.0000
Task 2	1.0000	1.0000	1.0000	1.0000	1.0000
Task 3	1.0000	1.0000	1.0000	1.0000	1.0000
Task 4	1.0000	1.0000	1.0000	1.0000	1.0000
Task 5	1.0000	1.0000	1.0000	1.0000	1.0000

	Station 1	Station 2	Station 3	Station 4	Station 5
Task 1	8.3561	0.87041	1.2190	1.2090	0.3487
Task 2	0.3487	1.7172	9.0760	0.5127	0.3487
Task 3	0.8144	8.7183	1.0109	1.1110	0.3487
Task 4	10.1428	0.8145	0.3487	0.3487	0.3487
Task 5	2.6011	7.6939	1.0109	0.3487	0.3487

Fig. 4. (a) The task allocations during three tours and (b) trail accumulation at the end of 15th tour (first matrix gives the initial trail accumulation).

tasks have higher probabilities of getting assigned to the stations. Fig. 4b also depicts how fast the trail value accumulation is; in only 15 tours, the values for some of the elements are sufficiently large to get differentiated.

3.3. Proposed algorithms

In Section 3.3.1, we present the proposed ant algorithm called ACS with random search (ACSm). Later, in Section 3.2.2, this algorithm is extended to a version called ACO to handle computational difficulties encountered in the original version.

3.3.1. ACS with random search (ACSm)

The proposed algorithm has all the standard features of ant algorithms (i.e., tour construction, local and global pheromone trial update, etc.) and some additional characteristics that enhance the performance of the algorithm. These additional characteristics are: (i) ACSm uses a more greedy station selection rule than that of AS and its variants, (ii) the global pheromone update rule is

only applied for the global-best solution (after a tour), and (iii) a local pheromone updating rule is applied (during a tour), while agents construct a solution. As briefly illustrated in Fig. 5, the detailed description of the features of the proposed algorithm is as follows:

Tour construction: In ACSm, an agent allocates task s to station j by applying the state transition rule given in Eq. (2) where q is a uniformly distributed random number in [0,1], q_0 is a parameter ($0 \leqslant q_0 \leqslant 1$), and S is a random variable selected according to the *random-proportional rule* given in Eq. (1).

$$s = \begin{cases} \arg \max\limits_{u \in allowed_u} \{[T_{uj}(t)][\eta_u]^{\beta}\} & \text{if } q \leqslant q_0 \text{ (exploration)} \\ S & \text{otherwise (exploitation)} \end{cases} \tag{2}$$

This state transition rule is the combination of Eqs. (1) and (2) and is called *pseudo-random-proportional rule* that favors allocations with large pheromone. The parameter q_0 controls the relative importance of exploitation versus exploration. For large q_0, the selection favors the exploration of the search space. In case of $q_0 = 1$, only the best task is chosen. If $q > q_0$, a task is chosen according to random-proportional rule that favors the exploitation of the search space using Eq. (1). Note that the possible values of S are the remaining assignable and available tasks. Eq. (1) provides the probability of assigning task i to station j; when the probabilities are determined for all the assignable and available tasks, the selection is done by generating a $U(0,1)$ random variable and selecting the appropriate task whose $p_{ij}^k(t)$ value corresponds to the value of the $U(0,1)$ random variable.

Basics of pheromone trail update: During the partial solution construction and after the completion of a full solution at the end of a tour, some of the pheromone trails are reduced by a constant factor ρ_i (pheromone evaporation rate, $0 < \rho_i < 1$) and some other pheromone trails are reinforced by a value. Rate ρ_i limits the uncontrolled accumulation of the pheromone trails and enables the algorithm to ignore previously made poor selections. The amount of pheromone deposited by each agent is proportional to the quality of this agent's solution (i.e., number of stations).

Global pheromone trail update: In ACSm only the globally best agent, which finds the best solution from the beginning is allowed to deposit pheromone. The global updating is performed after a tour is completed according to Eq. (3):

$$T_{ij}(t) = (1 - \rho_1) T_{ij}(t-1) + \rho_1 \cdot \Delta T_{ij}^{(gb)}(t) \tag{3}$$

where

$$\Delta T_{ij}^{(gb)}(t) = \begin{cases} \dfrac{Q}{f^{(gb)}(t)} & \text{if } (i,j) \text{ is part of the global best solution} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$0 \leqslant \rho_1 \leqslant 1$ is pheromone evaporation parameter, Q is a constant, and $f^{(gb)}(t)$ is the station number of the up-to-now best solution. In traditional ant algorithms (the variants of AS, not ACS), all trail values of task allocations receive a reinforcement as long as they are part of a solution. This reinforcement is directly proportional to (i) how many agents use this allocation as a part of their solution and (ii) the quality of this solution. A shortcoming of this procedure is pointed out by Bullnheimer et al. (1999) as follows. During the search, if the overall solution quality rises and the difference between the solutions decreases, the distinctive mark (emphasis) on better solutions diminishes. Consequently, the difference in trail values and the selection probabilities will decrease. Therefore, the exploitation of the solution space will not be as high as originally desired. However, only those trail values that belong to the up-to-now best solution receive reinforcement. Trail update mechanism directs the search towards better solutions by allowing the pheromone accumulation to be affected by the quality of the best solution (not all solutions).

Local pheromone trail update: In ACSm, ants use a local update rule in addition to the global updating rule. In other words, following the allocation of task i to station j, the pheromone level is updated using Eq. (5):

$$T_{ij}(t) = (1 - \rho_2) T_{ij}(t-1) + \rho_2 \cdot \tau_0 \tag{5}$$

where ρ_2, $0 \leqslant \rho_2 \leqslant 1$, is pheromone evaporation parameter, and τ_0 is a very small amount of pheromone level that controls the amount of pheromone reinforcement of the related trail values. The effect of the local update is to change the desirability of tasks dynamically. The local updating rule makes already chosen tasks less desirable for the forthcoming selections. By this way, the exploration of unallocated tasks is increased and the agents make a better use of pheromone information.

During our experimental studies with ACS, we investigated the task allocations and trail accumulation matrix entries in detail. Computational experiments indicate that after a short period, most trail matrix entries approach to zero. Since the selection probabilities are directly related to trail values, the information gained by pheromone accumulation is lost due to relative difference between high and low trail values. We suggest the following modifications to magnify the effect of trail values.

Modification step 1: We use a *secondary pheromone trail accumulation mechanism* (*SPTAM2*) to better analyze the trail accumulations. Following each tour, *SPTAM2* updates the related entry of a secondary trail matrix ($T2$) by adding the value 1 as follows:

$$T2_{ij'}(t) \leftarrow T2_{ij'}(t) + \sum_{k=1}^{m} \Delta T2_{ij'}^{k}(t) \quad \forall (i,j) \tag{6}$$

$$\Delta T2_{ij'}^{k}(t) = \begin{cases} 1 & \text{if task } i \text{ is allocated to location} \\ & \quad j' \text{ by globally best ant } k \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where m is the total number of agents used. During the experimental studies, with this modification, ACSm find the optimal station number in more test problems compared to our trials with ant algorithms. This supports our claim that, the information collected in $T2$ matrix is useful. However, we also note that this modification may lead to a $T2$ matrix with very small and large entry values.

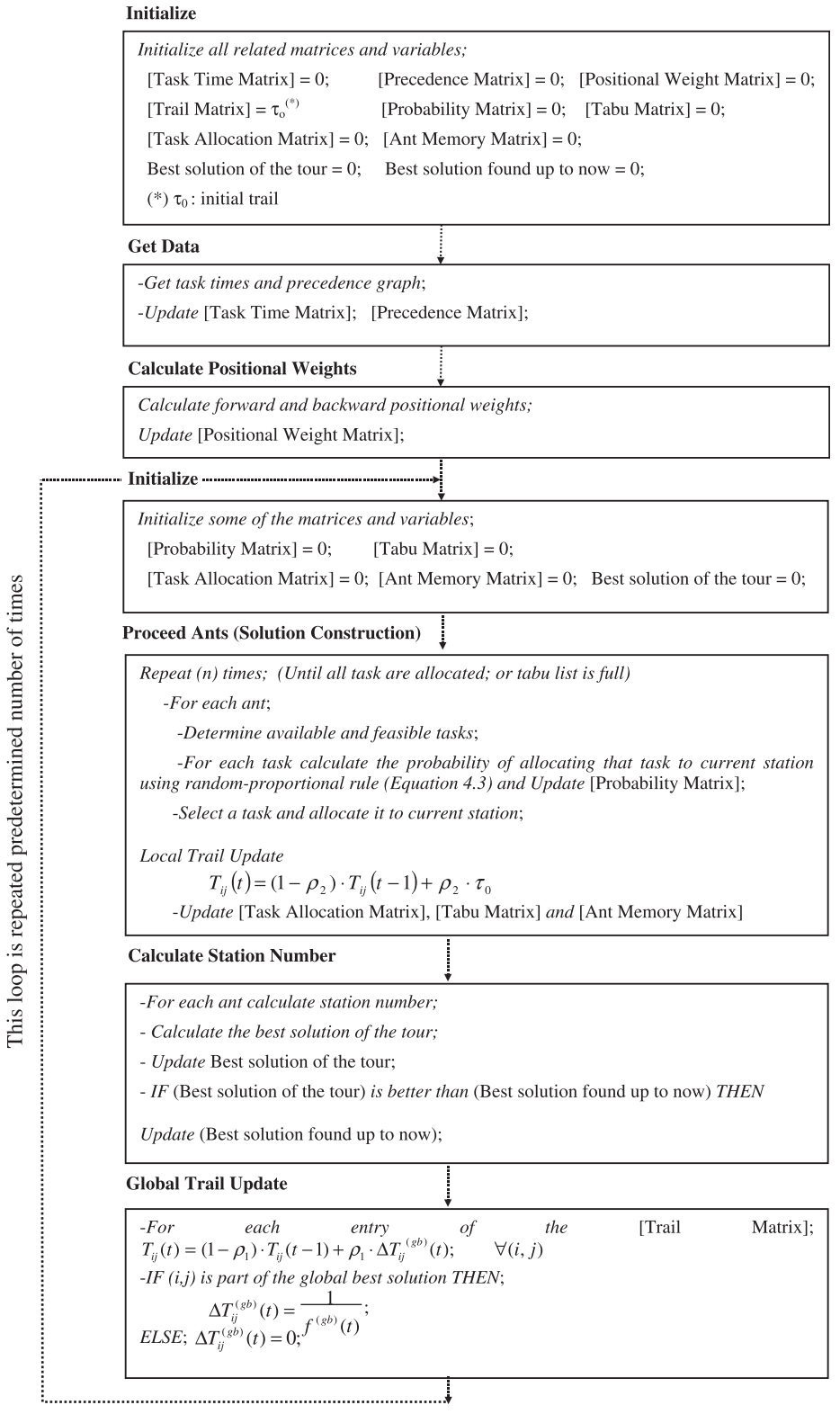

Fig. 5. Flowchart of Ant Colony System.

As stated above, this is not a desirable situation. Eventually, this causes the algorithm to consider only very large values (not any small values) and hence all the information accumulated in the matrix is not fully utilized. We overcome this problem using the following modification.

Modification step 2: In order to avoid over-accumulation, we update $T2$ only once in a tour for the same solutions which are generated by different agents (instead of updating the matrix for each agent). Hence, the adverse effects of over-emphasis or under-emphasis of some matrix entries are avoided. This modification requires keeping the best solutions in the memory. For that purpose we develop an efficient solution labeling mechanism. (See Nourie and Venta, 1991 for other possible solution labeling mechanisms.)

3.3.2. ACO method

We develop ACO to alleviate the computational difficulties encountered during the construction of $T2$ matrix especially for large size problems. Two versions are developed.

Version 1: We modify the local pheromone update mechanism of ACS not to lose the valuable information gained by the pheromone accumulation. The new local pheromone update mechanism is defined as follows:

$$T_{ij}(t) = (\rho_2)T_{ij}(t-1) + 1 \tag{8}$$

where ρ_2, $0 \leqslant \rho_2 \leqslant 1$. Instead of adding a small value $\rho_2 \cdot \tau_o$ (Eq. (5)), we reinforce the trail matrix by adding up 1 as done previously to update the $T2$ matrix. Thus, the related trail values are emphasized and their effects are magnified. For the pheromone evaporation, we use only ρ_2 as a multiplier instead of $(1-\rho_2)$. We also modify the global pheromone update mechanism of ACSm to reinforce the solutions with less number of stations. This is similar to the elitist strategy and the idea of discrimination of the solutions discussed in Hertz and Widmer (2003):

$$T_{ij}(t) = (1-\rho_1)T_{ij}(t-1) + \Delta T_{ij}^{(gb)}(t) \tag{9}$$

where

$$\Delta T_{ij}^{(gb)}(t) = \begin{cases} \dfrac{Q \cdot Optimal}{f^{(gb)}(t)} & \text{if } (i,j) \text{ is part of the global best solution} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$0 \leqslant \rho_1 \leqslant 1$ is pheromone evaporation parameter, *Optimal* is the optimal number of stations of the considered problem, $f^{(gb)}(t)$ is the station number of the up-to-now best solution and $Q \cdot Optimal/f^{(gb)}(t)$ is a *reward function* that satisfies more reinforcement for solutions with less number of stations. One can say that, Eq. (10) can be expressed by Eq. (4) by changing the parameter Q without any need for the multiplier '*Optimal*'. In Eq. (10) the influence of $\Delta T_{ij}^{(gb)}(t)$ on the pheromone accumulation is more dynamic compared to Eq. (4). The value of '*Optimal*' changes among the tour as the search proceeds, whereas the value Q is set up once at the very beginning. Moreover, Eq. (9) differs from Eq. (3) in terms of the magnitude of $\Delta T_{ij}^{(gb)}(t)$. Note that after the regular global pheromone update (as defined in Eqs. (9) and (10)), a secondary pheromone update is done for the globally best agent.

When we examine the optimal task allocations of the experiments conducted by Scholl and Klein (1999), we note that most stations have zero idle times. We call these stations as *full-loaded stations*. Using this characteristic, we modify the state transition rule to obtain full-loaded stations. Specifically, we modify the state transition rule given in Eq. (2) to emphasize the tasks with large processing times. This increases the likelihood of assignment of these tasks and it becomes possible to allocate these tasks at the preliminary stages. Later, it is much easier to allocate the remaining tasks with small processing times to the remaining slots in the stations. The new random-proportional rule is defined as follows:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[T_{ij}(t)]^{\alpha}[\eta_i]^{\beta}[t_i]^{\beta}}{\sum_{z \in allowed_k}([T_{zj}(t)]^{\alpha}[\eta_z]^{\beta}[t_i]^{\beta})} & \text{if } i \in allowed_k \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where $allowed_k$ is the set of available and assignable tasks for agent k. An agent allocates task s to station j by applying the *new pseudo-random-proportional rule* given by Eq. (12):

$$s = \begin{cases} \arg \max_{u \in allowed_u} \{[T_{uj}(t)][\eta_u]^{\beta}[t_u]^{\beta}\} & \text{if } q \leqslant q_0 \\ S & \text{otherwise} \end{cases} \tag{12}$$

where q is a random variable in [0,1], q_0 is a parameter ($0 \leqslant q_0 \leqslant 1$), and S is a random variable selected according to the rule given in Eq. (11).

Version 2 is the extension of *Version* 1 with *a secondary global pheromone trail update mechanism*. This modified version is proposed based on our observations on the insufficiency of the local update mechanism in two large size problems. During the solution of these large problems, most of the entries in the trail matrix become almost equal to each other that prevent the selection mechanism distinguish better tasks. We modify the local pheromone update mechanism of *Version* 1 as follows:

$$T'_{ij}(t) = (\rho_2)T'_{ij}(t-1) \tag{13}$$

We also modify the global pheromone update mechanism to emphasize allocations which yield full-loaded stations. The trail values of the tasks belonging to full-loaded stations are reinforced with a high number using Eq. (15). Thus, the trail values of these tasks will be distinguished easily after a few tours.

$$T'_{ij}(t) = T'_{ij}(t) + \Delta T_{ij}^{(gb-full\ load)}(t) \tag{14}$$

where

$$\Delta T_{ij}^{(gb-full\ load)}(t) = \begin{cases} Q_2 & \text{if } (i,j) \text{ is part of the global best solution and } j \text{ is } a \text{ full-loaded station} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

4. Experimental setting

The performance of the proposed algorithms are tested using the benchmark problems in the literature (Talbot et al., 1986; Hoffmann, 1990, 1992; Scholl, 1993). These data sets are available at www.assembly-line-balancing.de. In our study, two data sets are used: Talbot et al. (1986) data set with 64 instances of problem sizes ranging from 8 to 111 tasks and the data set of Scholl (1993) with 168 instances ranging from 25 to 297 tasks.

Number of tasks (n), task processing times, precedence relations, cycle time (C), number of ants (m), α, β, ρ_1, ρ_2, Q, Q_2, q_0, τ_0 are the input parameters of the algorithms. Number of tasks (n), task processing times, and precedence relations for each problem are given at the above website. The other parameters are discussed below.

Number of ants (m): Dorigo and Stützle (2000) suggest that ACO algorithms perform better when the number of ants is set to a value $m>1$. Dorigo et al. (1991a, b, 1996), Colorni et al. (1991, 1992) also suggest that the optimal number of ants should be taken close to the number of cities ($m\approx n$) for the TSP. In our implementation, we run pilot studies to test the effect of the number of ants on the system performance. A small size (Jackson's problem with 11 tasks and cycle time of 10), a medium size (Gunther's problem with 35 tasks and cycle time of 54) and a large size problem (Barthold's problem with 148 tasks and cycle time of 805) are chosen for this purpose. We evaluate the performance of proposed methods by changing the number of ants from 1 to $2n$, given (100 replications $\times$ 100 ant tours) for Jackson and Gunther problems and (1 replication $\times$ 100 ant tours) for the Barthold's problem. The results indicate that the proposed algorithms perform well when $m\approx n$. Usually for large problem instances, the number of ants are suggested to be less than 10 due to the excessive computation time. We also observe that our algorithm does not yield satisfactory results for $m<10$. Our pilot studies on *Version* 2 suggest to restrict ant number as $m\times(1/4)\approx ne$

We have also conducted a second set of experiments to see the performance of the proposed algorithms with respect to various values of α, β, ρ_1, ρ_2, Q, Q_2, q_0 and τ_0 using the test problems; specifically, $\alpha\in\{0,1,2,5\}$, $\beta\in\{0,1,2,5\}$, ρ_1 (also ρ_2)$\in\{0.1,0.4,0.7,0.9,0.99\}$, Q (also Q_2)$\in\{1,10,100,1000\}$, $q_0\in\{0,0.2,0.4,0.6,0.8,1\}$ and $\tau_0\in\{0.01,0.001,0.0028,0.003\}$. The value of τ_0 is suggested to be very small that depends on the problem size. Usually it is determined as $\tau_0=1/(f(s_h\times n)$, where s_h is a heuristic solution and τ_0 varies as $0.0001<\tau_0<0.0313$. As reported in Alp (2004), the results of the parameter optimization indicate that the number of stations is not sensitive to parameter values whereas the number of replications to reach the best solution is affected by different combinations even though these differences are small. Moreover, there is no best combination of these parameter values for all these algorithms (Alp, 2004). As a result the parameter values in Table 1 are suggested for each algorithm.

5. Computational results

In our pilot runs, we note that the traditional (or standard) ant algorithms are not successful to solve UALB due to the flat topology of the objective function. Hertz and Widmer (2003) has made similar observations that their search mechanism cannot escape from the large plateaus to find the valleys if the function is too flat. The objective function of UALBP unfortunately consists of vast plateaus (solutions with equal number of stations) and a few valleys (solutions with one less number of stations) cannot be reached during the update.

The computational results are presented in Table 2 for the benchmark problems. We first compare the proposed ant algorithms against the two best known algorithms in the literature: ULINO of Scholl and Klein (1999) and SA-based algorithm of Erel et al. (2001). The results of the experiments on 190 problem instances indicate that ULINO finds the optimal solution for 169 out of 190 instances. In six instances, the optimum is given in intervals (i.e., the best lower and upper bounds reported). In the remaining instances, the optimum could not be identified by ULINO. SA finds the optimal solution for 127 out of 187 instances. Among the proposed ant algorithms, the best performer is the ACO-Version 2 that finds the optimal solution for 144 instances (better than SA). In six instances, this new proposed algorithm finds upper and lower bounds that ULINO could not. An analysis of the results also indicates that ULINO performs better that ACO-Version 2 for only the largest size problem with 297 tasks. This suggests that efficient labeling and matrix update procedures may be needed to solve large size problems with ACO. Recall that ULINO is the best known constructive algorithm and SA is the best known iterative meta-heuristic developed for these problems. As a result, we conclude that ACO has the potential to be considered

Table 1
Best set of parameters for each proposed method.

Method	Parameters							
	α	β	ρ_1	ρ_2	Q	Q_2	q_0	τ_0
ACSm	1	1	0.4	0.4	1	–	0.2	0.0028
ACO method—Version 1	1	1	0.9 and 0.99[a]	0.9 and 0.99[a]	1	–	0.2 and 0.3[a]	–
ACO method—Version 2	1	1	0.99	0.99	1	100 and 1000[a]	0.8 and 0.3[a]	–

[a] Problems are solved at each level and the best is taken.

Table 2
Computational results.

	Optimal unknown	Optimal is found	Optimal is not found	Comparison (each row includes detailed comparison of performances of the proposed ant algorithm with SA and ULINO)	
				Number of instances	Explanation
ULINO	15	169	6		
SA-based heuristic	15	127	45		
ACSm	15	81	90[a]	9	ACSm is better than SA
			4[b]	58	SA is better than ACS
				2	None of them find optimal; SA finds one station more
				1	None of them find optimal; SA finds two stations more
				6	ULINO performs poor; ACSm finds upper bound for five instances
				88	ULINO performs better
ACO approach—Version 1	15	108	67[a]	25	Ver 1 finds optimal; SA finds one station more
				47	SA is better than Ver 1
				1	None of them find optimal; SA finds three stations more
				1	None of them find optimal; SA finds two stations more
				6	ULINO perfoms poor; Ver 1 finds upper bound
				61	ULINO performs better
ACO approach—Version 2	15	144	31[a]	39	Ver 2 finds optimal; SA finds one station more
				1	Ver 2 finds optimal; SA finds four stations more
				1	Ver 2 finds optimal; SA finds three stations more
				1	Ver 2 finds optimal; SA finds two stations more
				25	SA is better than Ver 2
				3	ULINO perfoms poor; Ver 2 finds lower bound
				3	ULINO perfoms poor; Ver 2 finds upper bound
				28	ULINO performs better

[a] The solution requires one additional station more than the optimal solution.
[b] The solution requires two additional stations more than the optimal solution.

as a serious alternative to solve UALBP. Besides, computational burden of the ant algorithm is reasonably small. The algorithm is written in Borland Delphi 6.0 and the average computational time is about a few seconds on an AMD Athlon XP 2000+, 266 Mhz machine with 256 MB RAM (333MHZ). Only 7 seconds are necessary to complete 250 agent tours with 90 agents for the largest problem. Scholl and Klein (1999) report that the average CPU time reported for ULINO to solve the 168 instances considered in their data set is 73.01 seconds (a time limit of 500 CPU seconds are given for each instance).

6. Conclusions and further research directions

In this study, we propose an ACO approach to solve the single-model U-type assembly line balancing problem (UALBP). In general, the proposed ACO algorithm outperforms the SA and displays very competitive performance against the state-of-the-art ULINO algorithm. For instance, we find upper and lower bounds of six instances that ULINO could not. These results are quite encouraging in the sense that researchers can further refine the proposed approach to solve the U-line assembly balancing problem. Other future research topics can be as follows:

(i) It would be interesting to augment the proposed ACO with meta-heuristics. But this augmentation must be done in such a way that the hybrid system can provide a fast search in the solution space.
(ii) A metaheuristic can also be used to fine-tune parameters. When there are many parameters, it is quite tedious to fine-tune these parameters using an experimental design. For example, Botee and Bonabeau (1998) use a GA to select some of the parameters of the ACO algorithm. This can be also be used for the proposed ant algorithms.
(iii) The ACO can also be used to solve type 2 problems (minimizing the cycle time given the number of stations). In this case, one would have to re-balance an existing design iteratively for a particular number of stations.
(iv) Finally, the proposed methods can be applied to different UALBP versions such as, stochastic assembly line balancing problems in which task times are

assumed to be random variables, mixed-model assembly systems, etc. It is possible to extend the proposed methods for more complex U-lines such as, multi-lines in a single U-line, double-dependent U-lines, embedded U-lines, and multi-U-line facilities.

References

Aase, G.R., Schniederjahn, M.J., Olson, J.R., 2003. U-OPT: an analysis of exact U-shaped line balancing procedures. International Journal of Production Research 41, 4185–4210.

Aase, G.R., Olson, J.R., Schniederjahn, M.J., 2004. U-shaped assembly line layouts and their impact on labor productivity: an experimental study. European Journal of Operational Research 156, 698–711.

Alp, A., 2004. Ant colony optimization for the single model U-type assembly line balancing problem. Unpublished MS Thesis, Department of Industrial Engineering, Bilkent University.

Bautista, J., Pereira, J., 2002. Ant algorithms for assembly line balancing. In: Lecture Notes in Computer Science, vol. 2463, pp. 65–75.

Bautista, J., Pereira, J., 2007. Ant algorithms for a time and space constrained assembly line balancing problem. European Journal of Operational Research 177, 2016–2032.

Besten, M., Stützle, T., Dorigo, M., 2000. Ant colony optimization for the total weighted tardiness problem. In: Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J., Schwefel, H.P. (Eds.), Parallel Problem Solving from Nature: Sixth International Conference. Lecture Notes in Computer Science, vol. 1917. Springer, Berlin, pp. 611–620 (also available as Technical Report IRIDIA/99-16. Université Libre de Bruxelles, Belgium).

Besten, M.L., Stützle, T., Dorigo, M., 2000. An ant colony optimization application to the single machine total weighted tardiness problem. In: Dorigo, M., Middendorf, M., Stützle, T. (Eds.), Abstract Proceedings of ANTS'2000: From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, Brussels, Belgium, pp. 39–42.

Blum, C., Sampels, M., 2002. Ant colony optimization for fop shop scheduling: a case study on different pheromone representations. In: Proceedings of 2002 Congress on Evolutionary Computation (CEC '02), vol. 2. IEEE Computer Society Press, pp. 1558–1563.

Botee, H.M., Bonabeau, E., 1998. Evolving ant colony optimization. Advance Complex Systems 1, 149–159.

Boysen, N., Fliedner, M., 2008. A versatile algorithm for assembly line balancing. European Journal of Operational Research 184, 39–56.

Bullnheimer, B., Hartl, R.F., Strauss, C., 1999. A new rank-based version of the ant system: a computational study. Central European Journal for Operations Research and Economics 7 (1), 25–38.

Chiang, W.C., Urban, T.L., 2006. The stochastic U-line balancing problem: a heuristic procedure. European Journal of Operational Research 175, 1767–1781.

Colorni, A., Dorigo, M., Maniezzo, V., 1992. An investigation of some properties of an ant algorithm. In: Manner, R., Manderick, B. (Eds.), Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92), Brussels, Belgium. Elsevier, Amsterdam, pp. 509–520.

Colorni, A., Dorigo, M., Maniezzo, V., 1991. Distributed optimization by ant colonies. In: Proceedings of ECAL91-European Conference on Artificial Life, Paris, France. Elsevier, Amsterdam, pp. 134–142.

Dorigo, M., Blum, C., 2005. Ant colony optimization theory: a survey. Theoretical Computer Science 344 (2–3), 243–278.

Dorigo, M., Gambardella, L.M., 1996. A study of some properties of Ant-Q. In: Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.S. (Eds.), Proceedings of PPSN IV—Fourth International Conference on Parallel Problem Solving From Nature. Springer, Berlin, pp. 656–665.

Dorigo, M., Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the travelling salesman problem. IEEE Transactions on Evolutionary Computation 1 (1), 53–66.

Dorigo, M., Gambardella, L.M., 1997. Ant colonies for the traveling salesman problem. BioSystems 43, 73–81.

Dorigo, M., Maniezzo, V., Colorni, A., 1991a. Ant system: an autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartmentio di Eletronica e Informazione, Politecnico di Mileno, Italy.

Dorigo, M., Maniezzo, V., Colorni, A., 1991b. Positive feedback as a search strategy. Technical Report 91-016, Dipartmentio di Eletronica, Politecnico di Mileno, Italy.

Dorigo, M., Maniezzo, V., Colorni, A., 1996. The ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics—Part B 26 (1), 1–13.

Dorigo, M., Socha, K., 2007. An introduction to ant colony optimization. In: Gonzalez, T.F. (Ed.), Handbook Approximation Algorithms and Metaheuristics. Chapman and Hall/CRC Press.

Dorigo, M., Stützle, T., 2000. The ant colony optimization metaheuristic: algorithms, applications, and advances. Technical Report, IRIDIA/2000-32, IRIDIA, University Libre de Bruxelles, Belgium.

Dorigo, M., Stützle, T., 2004. Ant Colony Optimization. MIT Press, Cambridge, MA.

Erel, E., Sabuncuoglu, I., Aksu, B.A., 2001. Balancing of U-type assembly systems using simulated annealing. International Journal of Production Research 39, 3003–3015.

Fenet, S., Hassas, S., 2000. A.N.T.: a distributed problem-solving framework based on mobile agents. In: Lasker, G.E., Dospisil, J., Kendall, E., Kendall, (Eds.), Advances in Mobile Agents Systems Research, vol. 1. Theory and Practice, MAA'2000, International Institute for Advanced Studies in Systems Research and Cybernetics, pp. 39–44.

Gagné, C., Gravel, M., Price, W.L., 2001. A look-ahead addition to the ant colony optimization algorithm and its application to an industrial scheduling problem. In: Proceedings of 4th Metaheuristics International Conference (MIC'2001), Porto, Portugal, pp. 79–84.

Gajpal, Y., Rajendran, C., 2006. An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. International Journal of Production Economics 101, 259–272.

Gambardella, L.M., Dorigo, M., 1995. Ant-Q: a reinforcement learning approach to the travelling salesman problem. In: Prieditis, A., Russell, S. (Eds.), Proceedings of 12th International Conference on Machine Learning (ML-95). Morgan Kaufmann, Palo Alto, CA, pp. 252–260.

Guerriero, F., Miltenburg, J., 2003. The stochastic U-line balancing problem. Naval Research Logistics 50, 31–57.

Hertz, A., Widmer, M., 2003. Guidelines for the use of meta-heuristics in combinatorial optimization. European Journal of Operational Research 151, 247–252.

Hoffmann, T.R., 1990. Assembly line balancing—a set of challenging problems. International Journal of Production Research 28, 1807–1815.

Hoffmann, T.R., 1992. EUREKA: a hybrid system for assembly line balancing. Management Science 38, 39–42.

Holland, J.F., 1975. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1989. Optimization by simulated annealing: an experimental evaluation; Part I, graph partitioning. Operations Research 37, 865–892.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1991. Optimization by simulated annealing: an experimental evaluation. Part II: graph coloring and number partitioning. Operations Research 39, 378–406.

Kara, Y., Ozcan, U., Peker, A., 2007. Balancing and sequencing mixed-model just-in-time U-lines with multiple objectives. Applied Mathematics and Computation 184, 566–588.

Kim, Y.K., Kim, S.J., Kim, J.Y., 2000. Balancing and sequencing mixed-model U-lines with a co-evolutionary algorithm. Production Planning & Control 11 (8), 754–764.

Krishnaiyer, K., Cheraghi, S.H., 2002. Ant algorithms: review and future applications. In: Proceedings of IERC'02, Industrial Engineering Research Conference, Orlando, FL, USA.

Lin, B.M.T., Lu, C.Y., Shyu, S.J., Tsai, C.Y., 2008. Development of new features of ant colony optimization for flowshop scheduling. International Journal of Production Economics 112, 742–755.

Maniezzo, V., Carbonaro, A., 2001. Ant colony optimization: an overview. In: Ribeiro, C. (Ed.), Essays and Surveys in Metaheuristics. Kluwer Academic Press, Dordrecht, pp. 21–44.

McMullen, P.R., Tarasewich, P., 2003. Using ant techniques to solve the assembly line balancing problem. IIE Transactions 35, 605–617.

Middendorf, M., Reischle, F., Schmeck, H., 2002. Multi colony ant algorithms. Journal of Heuristics, 305–320.

Miltenburg, G.J., Sparling, D., 1995. Optimal solution algorithms for the U-line balancing problem. Working Paper, McMaster University, Hamilton, Ontario, Canada.

Miltenburg, G.J., Wijngaard, J., 1994. The U-line line balancing problem. Management Science 40 (10), 1378–1388.

Miltenburg, J., 1998. Balancing U-lines in a multiple U-line facility. European Journal of Operational Research 109, 1–23.

Miltenburg, J., 2001. One-piece flow manufacturing on U-shaped production lines: a tutorial. IIE Transactions 33, 303–321.

Miltenburg, J., 2000. The effect of breakdowns on U-shaped production lines. International Journal of Production Research 38, 353–364.

Montgomery, J., Randall, M., 2002. Alternative pheromone applications for ant colony optimization, Technical Report TR02-07, School of Information Technology, Bond University, Australia.

Nakade, K., Ohno, K., 1999. An optimal worker allocation problem for a U-shaped production line. International Journal of Production Economics 60 (1), 353–358.

Nakade, K., Ohno, K., 2003. Separate and carousel type allocations of workers in a U-shaped production line. European Journal of Operational Research 145, 403–424.

Nourie, F.J., Venta, E.R., 1991. Finding optimal line balances with OptPack. Operations Research Letter 10, 165–171.

Scholl, A., 1993. Data of assembly line balancing problems. Schriften zur Quantitativen Betriebswirtschaftslehre 16/93, TU Darmstadt.

Scholl, A., Klein, R., 1999. ULINO: optimally balancing U-shaped JIT assembly lines. International Journal of Production Research 37 (4), 721–736.

Sparling, D., 1998. Balancing just-in-time production units: the N U-line balancing problem. Information Systems and Operational Research 36 (4), 215–237.

Sparling, D., Miltenburg, J., 1998. The mixed-model U-line balancing problem. International Journal of Production Research 36 (2), 485–501.

Stützle, T., 1998. An ant approach to the flow shop problem. In: Proceedings of the Sixth European Congress on Intelligent Techniques and Soft Computing (EU-FIT'98), Verlag Mainz, Aachen, 3, pp. 1560–1564.

Stützle, T., Dorigo, M., 1999. ACO algorithms for the travelling salesman problem. In: Miettinen, K., Makela, M., Neittaanmaki, P., Periaux, J. (Eds.), Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications. Wiley, New York, pp. 163–183.

T'kindt, V., Monmarché, N., Tercinet, F., Laügt, D., 2002. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. European Journal of Operational Research 142, 250–257.

Talbot, F.B., Patterson, J.H., Gehrlein, W.V., 1986. A comparative evaluation of heuristic line balancing techniques. Management Science 32, 430–454.

Urban, T.L., 1998. Optimal balancing of U-shaped assembly lines. Management Science 44, 738–741.

Urban, T.L., Chiang, W.C., 2006. An optimal piecewise-linear program for the U-line balancing problem with stochastic task times. European Journal of Operational Research 168, 771–782.