Beam-ACO applied to assembly line balancing*

Christian Blum[1], Joaquín Bautista[2], and Jordi Pereira[3]

[1] ALBCOM, Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, E-08034 Barcelona, Spain
cblum@lsi.upc.edu

[2] ETSEIB, Nissan Chair
Universitat Politècnica de Catalunya, E-08028 Barcelona, Spain
joaquin.bautista@upc.es

[3]ETSEIB, Dept. d'Organitzacíó d'Empreses
Universitat Politècnica de Catalunya, E-08028 Barcelona, Spain
jorge.pereira@upc.edu

May 15, 2006

Abstract

Assembly line balancing concerns the design of assembly lines for the manufacturing of products. In this paper we consider the time and space constrained simple assembly line balancing problem with the objective of minimizing the number of necessary work stations. This problem is denoted by TSALBP-1 in the literature. For tackling this problem we propose a Beam-ACO approach, which is an algorithm that results from hybridizing ant colony optimization with beam search. The experimental results show that our algorithm is a state-of-the-art metaheuristic for this problem.

1 Introduction

Simple assembly line balancing (SALBP) [14] concerns the manufacturing of products via assembly lines. An assembly line consists of a set of work stations arranged in a line, and by a transport system which moves the product to be manufactured along the line. The product is manufactured by performing a given set of tasks $T = \{1, \ldots, n\}$. Each of the tasks $j \in T$ has a processing time $t_j > 0$. A solution to a SALBP instance is obtained by the assignment of all tasks to work stations subject to precedence constraints between the tasks. The assembly line moves in constant speed, which leads to a maximum time c (called cyle time) in which the tasks assigned to each work station must be performed. Therefore, an assignment of tasks to work stations is only valid if the work load of each work station does not exceed the cycle

*This work was supported by grants TIN-2005-08818-C04-01 and DPI2004-03475 of the Spanish government, and by the "Juan de la Cierva" program of the Spanish Ministry of Science and Technology of which Christian Blum is a post-doctoral research fellow. Moreover, we acknowledge Nissan Spain and the UPC Nissan Chair for partially funding this work as well as for providing real data.

time. Several different optimization objectives are possible. A wide-spread objective is the minimization of the number of necessary work stations. This particular problem is denoted by SALBP-1 in the literature. It can be seen as a bin packing problem with additional side constraints (see [1]).

The SALBP-1 problem has been one of the most studied assembly line problems in the last 40 years. Solution approaches include constructive heuristics based on priority rules (see [20]), enumerative procedures such as branch & bound approaches (see [10, 9, 15, 19]), and several metaheuristics such as tabu search [16, 11], simulated annealing [21], evolutionary computation [8], and ant colony optmization [2, 3]. The current situation for the SALBP-1 problem is quite unusual: Enumerative approaches such as SALOME [15] still appear to be at least as successful as metaheuristic approaches. However, the interest in well-working metaheuristics is high, because the applicability of enumerative methods is limited to academic formulations of the problem, and even slight differences between a real problem and the academic SALBP-1 make an enumerative approach unapplicable. In this work we consider a generalization of the SALBP-1 problem, called TSALBP-1 (the time and space constrained simple assembly line balancing problem with the objective of minimizing the number of necessary work stations). This generalization was proposed by Bautista and Pereira in [3]. The generalization consists in adding space constraints to the processing time constraints, and was motived by a real assembly line balancing problem at the Nissan plant in Barcelona, Spain.

Motivation. Many existing metaheuristic techniques for the SALBP-1 problem do not employ a direct solution approach. They rather solve the SALBP-1 in the following indirect way: Given an initial solution with m work stations, the metaheuristic is applied to find a solution with a fixed number of $m-1$ work stations with a cycle time c' as low as possible (that is, the cycle time is considered variable). If a solution can be found with $c' \leq c$, the found solution is a valid solution for SALBP-1 with $m-1$ work stations. In the next step, the number of work stations is reduced by one, and the metaheuristic is applied again. The existing ACO approach for SALBP-1 (see [3]) works in this way.

In this work we wanted to study if a clever ACO hybrid can be directly applied to solve the SALBP-1 (and its generalization, the TSALBP-1) with the goal of improving over the indirect approaches. In order to achieve this, we tackle the TSALBP-1 with an hybrid ant colony optimization (ACO) [6] algorithm called Beam-ACO [4], which is obtained by hybridizing ACO with beam search.[1] The main idea of Beam-ACO is to allow the extension of partial solutions in several different ways. A lower bound is used to limit the number of partial solutions that are visited by the algorithm. The existence of an accurate—and computationally inexpensive—lower bound is crucial for the success of beam search (respectively, Beam-ACO). In Beam-ACO, artificial ants perform a probabilistic beam search in which the extension of partial solutions is done in the ACO fashion rather than deterministically.

Paper outline. In Section 2 we present a technical definition of the TSALBP-1 problem. In Section 3 we first study the potential of parametrizing a simple priority rule based heuristic, before we outline Beam-ACO. Finally, in Section 4 we present the computational results, and in Section 5 we offer conclusions and an outlook to the future.

[1] Beam search is a classical tree search method that was introduced in the context of scheduling [13].

2 TSALBP-1

An instance (T, G, c, a) of the TSALBP-1 problem consists of four components. $T = \{1, \ldots, n\}$ is a set of n tasks that must be procecced by a line of work stations. Each task $j \in T$ has a processing time $t_j > 0$, and a space requirement $a_j > 0$ (both values may be integer or real values). Furthermore, given is a precedence graph $G = (T, A)$, which is a directed graph without cycles whose nodes are the tasks. An arc $l_{i,j} \in A$ indicates that i must be processed before j. Given a task $j \in T$, we denote by $\mathrm{Pre}_j \subset T$ the set of tasks that must be processed before j. Finally, c is the processing time limit of a work station (called the cycle time), and a is the available space of a work station.

A solution is obtained by assigning each task to exactly one work station. In this work we represent a solution s as an ordered set $\langle S_1, \ldots, S_m \rangle$ of $m \leq n$ work stations S_k. Each work station $S_k \subseteq T$ is a set of tasks. A solution s is valid if the following conditions are fullfilled:

1. $\bigcup_{k=1}^{m} S_k = \{1, \ldots, n\}$ and $\bigcap_{k=1}^{m} S_k = \emptyset$. These conditions ensure that each task is assigned to exactly one work station.
2. $\sum_{j \in S_k} t_j \leq c$, for $k = 1, \ldots, m$. This ensures that no work station has too much load.
3. $\sum_{j \in S_k} a_j \leq a$, for $k = 1, \ldots, m$. Herewith is ensured that the space limits of the work stations are not exceeded.
4. For each task j in a work station S_k it must hold that $\bigcup_{l=1}^{k} S_l$ contains Pre_j, which ensures that the precedence constraints between the tasks are respected.

All our algorithms exclusively generate valid solutions. Finally, note that each SALBP-1 instance can be transformed into a TSALBP-1 instance by setting $t_j := a_j \; \forall \; j \in T$, and by setting $a := c$.

Objective function. The objective function of TSALBP-1 is the number of work stations of a solution, which must be minimized. Given a solution s, this objective function is denoted by $c_1(s) = |s|$. However, this objective function contains large plateaus, that is, many different solutions will have the same objective function value. Therefore, we introduce a second criteria in order to distiguish between solutions with the same number of work stations. The second criteria concerns the remaining time and space in the last work station $S_m \in s$. We use the following notations:

$$t^{\mathrm{rem}}(s) := c - \sum_{j \in S_m} t_j \qquad \text{and} \qquad a^{\mathrm{rem}}(s) := a - \sum_{j \in S_m} a_j \tag{1}$$

Using these notations, the second criteria is defined as $c_2(s) = \frac{t^{\mathrm{rem}}(s)}{c} + \frac{a^{\mathrm{rem}}(s)}{a}$. Having c_1 and c_2 we can indirectly defined a new objective function $f(\cdot)$ as follows. Given $s \neq s'$,

$$f(s) < f(s') \Leftrightarrow c_1(s) < c_1(s') \quad \textbf{OR} \quad c_1(s) = c_1(s') \text{ and } c_2(s) < c_2(s') \; . \tag{2}$$

This means that—in the case of equality concerning the first criteria—a solution s is regarded as better than a solution s' if and only if more space and time is remaining in the last work station of solution s. Note that despite the fact that Beam-ACO uses the objective function $f(\cdot)$, we will present the results only in terms of the original objective function.

Algorithm 1 Priority-rule heuristic for TSALBP-1

1: **input:** An instance (T, G, c, a) of TSALBP-1
2: $k := 0$
3: **while** $T \neq \emptyset$ **do**
4: $\quad k := k + 1$
5: $\quad S_k \leftarrow$FillWorkStation(T, k) /* see Algorithm 2 */
6: $\quad T := T \setminus S_k$
7: **end while**
8: **output:** Solution $s = \langle S_1, \ldots, S_k \rangle$

Reverse problem instances [14]. Given a problem instance (T, G, c, a), the corresponding reverse problem instance (T, G^r, c, a) is obtained by inverting all the arcs in the precedence graph G. Each solution $s^r = \langle S_1, \ldots, S_m \rangle$ to the reverse problem instance (T, G^r, c, a) can be converted into a solution s to the original problem instance (T, G, c, a) by inverting the ordered list of tasks, that is, $s = \langle S_m, \ldots, S_1 \rangle$. We introduce this property, because it is known from the literature that the reverse problem instance may be easier to solve than the original one, or vice versa.

3 The algorithm

In this section we outline our implementation of Beam-ACO for TSALBP-1. In the first part of the section we deal with a parametrized version of the constructive heuristic that is the basis of the Beam-ACO approach. The second part of the section is concerned with the Beam-ACO approach itself.

3.1 A parametrized priority rule heuristic

Priority rule heuristics are successful constructive methods for solving scheduling-type problems such as assembly line balancing. Therefore, we chose this type of algorithm as constructive mechanism for Beam-ACO. First we outline the heuristic, and then we study possible improvements of this heuristic by means of parametrization [7]. The priority rule heuristic for assembly line balancing works as follows (see also [3]): The algorithm assigns tasks to work stations, starting from the first work station. At each step, the algorithm selects one of the available tasks (that is, tasks whose predecessors are already assigned to work stations) and assigns it to the current work station. This is done until no available task can be added to the current work station without violating the cycle time or the space limitation. When this situation occurs, the algorithm opens the next work station. This process is continued until all tasks are assigned to work stations. This heuristic is pseudo-coded in Algorithm 1.

When filling a work station, the successive choice of tasks is performed in function ChooseTask(T') (see Algorithm 2), which works as follows. Let $T^{\text{av}} \subseteq T'$ be the set of tasks such that $\text{Pre}_j \cap T' = \emptyset$ for all $j \in T^{\text{av}}$, that is, T^{av} is the set of available tasks. Moreover, let $T^{\text{sat}} \subseteq T^{\text{av}}$ be the set of available tasks such that $c_{\text{rem}} - t_j = 0$ or $a_{\text{rem}} - a_j = 0$. Henceforth, we call T^{sat} the set of saturating avaiable tasks, because they saturate (in terms of time and/or space) the current work station. If T^{sat} is non-empty, a task is chosen from T^{sat}, otherwise from T^{av}. The actual choice of a task is done by using so-called priority rules. In our work we

Algorithm 2 Function FillWorkStation(T, k)

input: A set T of tasks, and the index k of the work station to be filled
$T' := T$
$S_k := \emptyset$
$c_{\text{rem}} := c$
$a_{\text{rem}} := a$
while $T' \neq \emptyset$ **and** $\exists\, i \in T'$ **s.t.** $c_{\text{rem}} - t_i \geq 0$ and $a_{\text{rem}} - a_i \geq 0$ **do**
  $j \leftarrow$ChooseTask(T')
  $T' := T' \setminus \{j\}$
  $S_k := S_k \cup \{j\}$
  $c_{\text{rem}} := c_{\text{rem}} - t_j$
  $a_{\text{rem}} := a_{\text{rem}} - a_j$
end while
output: Filled work station S_k

employ a mixed rule that gives joint priority to the duration of a task, the space requirement of a task, and the total number of tasks succeeding the given task:

$$\eta_j = \frac{t_j}{c} + \frac{a_j}{a} + \frac{\left|\text{Suc}_j^{\text{all}}\right|}{\max_{1 \leq i \leq n} \left|\text{Suc}_i^{\text{all}}\right|} \quad , j \in T^{\text{sat}} \text{ (respectively } j \in T^{\text{av}}) \tag{3}$$

Hereby, $\text{Suc}_j^{\text{all}}$ denotes the set of all tasks that can be reached from j in the precedence graph G via a directed path. Function ChooseTask(T') is implemented such that the task that maximizes the priority rule is chosen. The use of priority rule heuristics is quite popular, because they are usually fast in execution and achieve reasonably good results. Accordingly, many different priority rules have been proposed. The problem of priority rules is their inflexibility, that is, each rule generally works well only for certain types of problem instances. This was our motivation for introducing more flexibility into the priority rule shown in equation (3) before using it as heuristic information in our Beam-ACO approach. This can be done by parametrizing the three terms of the priority rule (3) as follows:

$$\eta_j = \left(\kappa_1 \cdot \frac{t_j}{c}\right) + \left(\kappa_2 \cdot \frac{a_j}{a}\right) + \left(\kappa_3 \cdot \frac{\left|\text{Suc}_j^{\text{all}}\right|}{\max_{1 \leq i \leq n} \left|\text{Suc}_i^{\text{all}}\right|}\right) \quad , \tag{4}$$

where $\kappa_1, \kappa_2, \kappa_3 \in [-1, 1]$. Given a problem instance, the goal is naturally to find values for κ_1, κ_2, and κ_3 such that the parametrized priority rule works best. This is a difficult optimization problem itself, which may be solved by techniques for continuous optimization that do not require gradient information. Examples from the field of swarm intelligence are ACO and particle swarm optimization. In fact, we implemented both the $\text{ACO}_{\mathbf{R}}$ algorithm by Socha [18], and a standard particle swarm optimizer as proposed by Shi and Kennedy in [17]. However, surprisingly both optimization techniques are outperformed by a simple random search algorithm (henceforth denoted by RAND), which works as follows: At each iteration, a setting for the parameters κ_1, κ_2, and κ_3 is chosen uniformly at random from the parameter value space. Then, two solutions are constructed with this parameter set by Algorithm 1:

1. A solution to the original problem instance.

2. A solution to the reverse problem instance. This solution is subsequently converted into a solution to the original problem instance.

RAND proceeds until a computation time limit is reached, and outputs the best solution found together with the parameter set which produced this best solution (with respect to the objective function $f(\cdot)$).

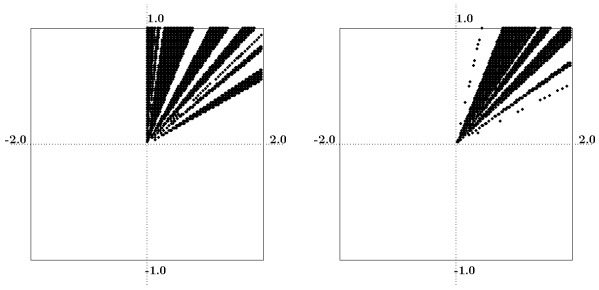
In order to test the potential of RAND, we applied the algorithm to all 269 problem instances from the SALBP-1 benchmark set by Scholl (see `http://www.assembly-line-balancing.de`). The results are shown in Table 1. The results of RAND (applied for 120 seconds to each problem instance) are compared to the results of the application of Algorithm 1 (applied to both, the original instance as well as the reverse instance). Algorithm 1 is applied with a parameter setting such that $\kappa_1 + \kappa_2 = 1.0$ and $\kappa_3 = 1.0$, which corresponds to the standard use of the priority rule as described in the literature. The second column of Table 1 indicates how many (out of 269 instances) were solved to optimality. The third table column gives the average gap to the optimal solution (for example, if the generated solution has 5 work stations, and the optimal solution has 3 work stations, the gap is 2). The fourth column provides the worst gap observed over the 269 instances, and the last column indicates the computation time (in seconds). In the case of Algorithm 1, the given time is the average computation time over the 269 instances. In the case of RAND, the given time is the average over the computation times needed to find the best solution in each of the 269 applications.

Table 1: Results of the application of Algorithm 1 (2nd table row) with a parameter setting such that $\kappa_1 + \kappa_2 = 1.0$ and $\kappa_3 = 1.0$, and of RAND to the 269 SALBP-1 instances from http://www.assembly-line-balancing.de.

	solved	**avgerage gap**	**worst gap**	**average time**
Algorithm 1	164	0.41	3	0.000017
RAND	200	0.26	2	0.0019

The results show the following. In a fraction of a second, RAND is able to find parameter settings which allow to solve 200 of the 269 problem instances. In contrast, with the standard parameter setting, only 164 instances can be solved. Also the average and worst gaps to the optimal solutions are reduced.

Finally, we wanted to study the reason for the effectiveness of a random search technique such as RAND. For this purpose we placed a fine grid over the parameter space, and applied Algorithm 1 (both to the original and the reverse instance) for each grid point. In Figure 1 we show some of our results in two dimensions. The x-axis shows $\kappa_1 + \kappa_2$, and the y-axis shows κ_3. A grid point is shown in black, if the correspsonding parameter setting leads to the generation of an optimal solution. The two cases that are shown in Figure 1 are typical for many instances of the benchmark set. Usually, the optimal parameter areas (shown in black) are large enough for a random search algorithm to quickly find an optimal parameter setting. Note that in Figure 1(a) is shown a case in which the standard setting does not lead to the construction of an optimal solution.



(a) $c = 3786$ (b) $c = 3985$

Figure 1: The black areas show the parameter settings with which optimal solutions are generated by Algorithm 1. The precedence graph used for the two figures is ARC83 from http://www.assembly-line-balancing.de.

3.2 Beam-ACO for TSALBP-1

Our Beam-ACO approach—pseudo-coded in Algorithm 3—works as follows. First, algorithm RAND (see previous section) is used in function DetermineHeuristicInformation() for setting the heuristic information. Then, the pheromone values are initialized. At each algorithm iteration, $a_{\rm o}$ ants construct solutions to the original problem instance, and $a_{\rm r}$ ants construct solutions to the reverse problem instance. The solutions to the reverse problem instance are subsequently converted to solutions to the original problem instance. During each solution construction a lower bound is used for detecting situations in which the resulting final solution must be worse than the best solution found by the algorithm so far. In this case the corresponding solution construction is aborted. Finally, the pheromone values are updated in function UpdatePheromoneTrail($\mathcal{T}$,$*$). The pheromone values are re-initilized in case of algorithm convergence. In Algorithm 3 we use the following notations: $\mathcal{T} = \{\tau_{j,k}\}_{j,k=1,\ldots,n}$ is the set of pheromone values. Hereby, a pheromone value $\tau_{j,k}$ represents the desirability of assigning task j to work station k. Furthermore, $s_{\rm ib}$ is the best solution constructed at an iteration, and $s_{\rm bsf}$ is the best solution found since the start of the algorithm. The functions of our algorithm are outlined in more detail below.

DetermineHeuristicInformation(): The parametrized priority rule heuristic outlined in Algorithm 1 is used as constructive mechansim of our Beam-ACO approach. In order to find good parameter settings for κ_1, κ_2, and κ_3 of Equation 4, this function applies algorithm RAND (see previous section) for 0.5 seconds.[2] Given these parameter values, the heuristic information is determined as follows. Let

$$\eta_{\min} := \min\{\eta_j \mid j \in T\} \quad (5)$$

$$\eta_{\max} := \max\{\eta_j \mid j \in T\} \ . \quad (6)$$

Then we defined the heuristic information for the Beam-ACO approach to be

$$\eta_j^{\rm aco} := \frac{\eta_j - \eta_{\min} + 1}{\eta_{\max}} \quad \forall\, j \in T \ . \quad (7)$$

[2]Note that this time limit is arbitrarily chosen. A smaller time limit (for example, 0.1) would probably work as well.

Algorithm 3 Beam-ACO for TSALBP-1

1: **input:** An instance (T, G, c, a) of TSALBP-1
2: $s_{\text{bsf}} \leftarrow$ DetermineHeuristicInformation()
3: **forall** $\tau_{j,k} \in \mathcal{T}$ **do** $\tau_{j,k} := 0.5$ **end forall**
4: $cf := 0$
5: **while** termination conditions not satisfied **do**
6: $I := \emptyset$
7: **for** $i = 1$ to a_{o} **do**
8: $s_i \leftarrow$ ConstructSolution($\mathcal{T}$) /* see Algorithm 4 */
9: **if** $s_i \neq$ NULL **then** $I := I \cup \{s_i\}$
10: **end for**
11: **for** $i = 1$ to a_{r} **do**
12: $s_i^r \leftarrow$ ConstructReverseSolution($\mathcal{T}$)
13: **if** $s_i^r \neq$ NULL **then**
14: Obtain a solution s_i to the original instance from s_i^r
15: $I := I \cup \{s_i\}$
16: **end if**
17: **end for**
18: $cf \leftarrow$ ComputeConvergenceFactor($\mathcal{T}$)
19: **if** $I = \emptyset$ **then**
20: UpdatePheromoneTrail($\mathcal{T}$,s_{bsf})
21: **else**
22: $s_{\text{ib}} := \min\{f(s_i) \mid s_i \in I\}$
23: UpdatePheromoneTrail($\mathcal{T}$,s_{ib})
24: **if** $f(s_{\text{ib}}) < f(s_{\text{bsf}})$ **then** $s_{\text{bsf}} := s_{\text{ib}}$
25: **end if**
26: **if** $cf < 0.05$ **then**
27: **forall** $\tau_{j,k} \in \mathcal{T}$ **do** $\tau_{j,k} := 0.5$ **end forall**
28: **end if**
29: **end while**
30: **output:** s_{bsf}

Note that we can not use the η_j values directly as heuristic information, because some of them might be negative. Additionally, the best solution found by RAND is returned by function DetermineHeuristicInformation(), and is used as the currently best found solution of the algorithm.

ConstructSolution($\mathcal{T}$): This function is pseudo-coded in Algorithm 4. An ant constructs a solution as follows. Work stations are successively filled one after the other. However, in contrast to Algorithm 1, an ant fills each work station in k_{ext} different ways (see lines 8-11 of Algorithm 4). In the context of beam search, k_{ext} is the number of extensions that may be obtained from a partial solution. For all our experiments we have used the setting of $k_{\text{ext}} = 50$.

In order to fill a work station, an ant uses the function FillWorkStation(T, k) which is pseudo-coded in Algorithm 2. Function ChooseTask(T') of Algorithm 2 is hereby implemented as follows. First, we flip a coin in order to decide if the construction step is performed deterministically, or probabilistically. In case of a deterministic construction step, the set of

tasks from which to choose an operation, denoted by T^{c}, is determined as follows: If the set of available saturating tasks T^{sat} is non-empty, we set $T^{\mathrm{c}} := T^{\mathrm{sat}}$, otherwise $T^{\mathrm{c}} := T^{\mathrm{av}}$ (see Section 2 for the definition of T^{sat} and T^{av}). Then, from T^{c} is chosen the task that maximizes

$$\mathbf{p}_j = \frac{\left(\sum_{i=1}^{k} \tau_{j,i}\right) \cdot \eta_j^{\mathrm{aco}}}{\sum_{l \in T^{\mathrm{c}}} \left(\sum_{i=1}^{k} \tau_{l,i}\right) \cdot \eta_l^{\mathrm{aco}}} \quad . \tag{8}$$

This formula uses the summation rule introduced by Merkle and Middendorf for scheduling problems [12]. In case of a probabilistic construction step, T^{c} is set to T^{av}, and a task is chosen by rhoulette-wheel-selection with respect to the probabilities shown in Equation 8.

After filling a work station, a lower bound $\mathrm{LB}(\cdot)$ is applied to the current partial solution s extended by the filled work station (see line 10 of Algorithm 4). The value of the lower bound indicates the minimum number of work stations needed by a solution that contains the current partial solution extended by the filled work station. In this work we used a very simple lower bound: Given a partial solution s, let T^{rem} be the set of tasks that are not yet assigned to work stations. Then:

$$\mathrm{LB}(s) = \max \left\{ \left\lceil \frac{\sum_{j \in T^{\mathrm{rem}}} t_j}{c} \right\rceil , \left\lceil \frac{\sum_{j \in T^{\mathrm{rem}}} a_j}{a} \right\rceil \right\} \tag{9}$$

Only if the lower bound value is not worse than the number of work stations of the best solution found so far, the extension of the current partial solution is considered (see line 10 of Algorithm 4). Finally, from all the possible extensions of a partial solution, the one with the lowest lower bound value is chosen (see line 13 of Algorithm 4). Note that this corresponds to a beam search algorithm with a beam width equal to 1. We decided for this setting, because in this work we only wanted to test the potential of a beam search approach. However, we want to make the reader aware of the fact that a proper setting of the beam width might improve the results of the algorithm even further.

Finally, function ConstructReverseSolution($\mathcal{T}$) works in the same way as function ConstructSolution($\mathcal{T}$), just that it constructs a solution for the reverse problem instance. The same pheromone values are used, just in a slightly different way. For example, the pheromone value that expresses the desirability to assign task j to the first work station of the reverse problem instance is pheromone value $\tau_{j,|s_{\mathrm{bsf}}|}$, instead of $\tau_{j,1}$, and so on.

ComputeConvergenceFactor($\mathcal{T}$): Given the current pheromone values, this function computes a value cf to indicate the state of convergence of the algorithm:

$$cf = 2 \cdot \left(\frac{\sum_{j=1}^{n} \sum_{k=1}^{|s_{\mathrm{bsf}}|} \min\{\tau_{\max} - \tau_{j,k}, \tau_{j,k} - \tau_{\min}\}}{n \cdot |s_{\mathrm{bsf}}| \cdot (\tau_{\max} - \tau_{\min})} \right) \tag{10}$$

When the pheromone values are initialized, cf is 1; on the other side, when the algorithm is converged, cf is 0. We have set $\tau_{\max}$ to 0.99, and $\tau_{\min}$ to 0.01. Note that the use of these

Algorithm 4 Function ConstructSolution($\mathcal{T}$,s_{bsf}) of Algorithm 3

input: The set of pheromone values $\mathcal{T}$, and s_{bsf}
$T = \{1, \ldots, n\}$
$k := 0$
$s := \langle\rangle$
while $T \neq \emptyset$ **do**
  $k := k + 1$
  $I := \emptyset$
  for $i = 1, \ldots, k_{\text{ext}}$ **do**
    $S_k^i =$FillWorkStation(T, k) /* see Algorithm 2 */
    if $\text{LB}(s \cup S_k^i) \leq |s_{\text{bsf}}|$ **then** $I = I \cup S_k^i$
  end for
  if $I \neq \emptyset$ **then**
    $S_k^* := \text{argmax}\{\text{LB}(s \cup S_k^i) \mid S_k^i \in I\}$
    $T := T \setminus S_k^*$
    $s := \langle S_1, \ldots, S_k^* \rangle$
  else
    output: NULL
  end if
end while
output: Solution $s = \langle S_1, \ldots, S_k \rangle$

bounds and their value setting is motivated by the implementation of $\mathcal{MAX}$-$\mathcal{MIN}$ AS algorithms implemented in the hyper-cube framework (see, for example, [5]).

UpdatePheromoneTrail($\mathcal{T}$,$*$): This function either uses solution s_{ib} or solution s_{bsf} for updating the pheromone values. s_{bsf} is only used in case no iteration best solution exists, due to solution construction abortions. Let us denote the updating solution by s_{upd}. Then, for $j = 1, \ldots, n$ and $k = 1, \ldots, |s_{\text{upd}}|$ the corresponding pheromone value $\tau_{j,k}$ is updated as follows:

$$\tau_{j,k} = \min\left\{\max\left\{\tau_{\min}, \tau_{j,k} + \rho \cdot (\delta_{j,k} - \tau_{j,k})\right\}, \tau_{\max}\right\} \quad , \tag{11}$$

where $\rho \in (0, 1]$ is a learning rate (which we have set to 0.1 for all the experiments). Moreover, $\delta_{j,k}$ is 1, if task j is assigned to work station k in solution s_{upd}, and 0 otherwise.

This concludes the description of our algorithm. The experimental results are outlined in the following section.

4 Computational results

We implemented the Beam-ACO algorithm in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with Intel Pentium 4 processor (3.06 GHz) and 1 Gb of memory. We performed three series of computational tests, which are outlined in the following.

Table 2: Results obtained by Beam-ACO in comparison to the results of solution techniques from the literature. The second table row provides the number of solved SALBP-1 instances (out of 269). The third table row contains the average computation times (in seconds) for finding the best solution of each run. Note that the computation time comparison is not really useful, because the computers that were used are quite different.

	SALOME [15]	**PrioTabu** [16]	**EurTabu** [16]	**ANTS** [3]	**HGA** [8]	**Beam-ACO**
solved	227	200	214	227	214	245
avg. time	98.6	101.8	62.6	13.84	n. g.	1.92

4.1 Results for SALBP-1 instances

First we applied Beam-ACO to all 269 SALBP-1 instances from the benchmark obtainable from `http://www.assembly-line-balancing.de`. The results of Beam-ACO are presented in a summarized form and compared to other approaches in Table 2. Beam-ACO was applied 10 times for 120 seconds to each problem instance. We can note that Beam-ACO solves more problem instances to optimality than any other available technique. In particular, we can note that Beam-ACO solves more instances than ANTS, which is an ACO algorithm that utilizes the indirect resolution approach as outlined in the introduction.

Exemplary we show the results of Beam-ACO for the 26 difficult instances based on the precedence graph called SCHOLL in Table 3. The results show that Beam-ACO can solve more instances to optimaliy (namely, 9) than the two most recent metaheuristic approaches. The computation times show that, in case the optimal solution can be found, this is usally the case after 30 to 40 seconds. In case the optimal solution is not found, the gap is never greater than 1, and the computation times are very low. This means that Beam-ACO finds very easily near-optimal solutions.

4.2 Results for the TSALBP-1 instances

We also applied our algorithm to the 269 TSALBP-1 instances that were generated by Bautista and Pereira (see [3]) from the 269 SALBP-1 instances.[3] We exemplary show the results concerning the 26 instances based on the precedence graph called SCHOLL in Table 4. The results are compared to the results of the standard ACO approach ANTS by Bautista and Pereira (see [3]), which is the only available technique for TSALBP-1. The results show that in 14 out of 26 cases, Beam-ACO improves the best known solution. In other 11 cases, Beam-ACO matches the best known solution values. Only in one case Beam-ACO does not match the performance of ANTS. This case is characterized by a relatively small cycle time. In general, we noticed the tendency that Beam-ACO—when applied to TSALBP-1 instances—is better when the cycle time is bigger. This might be caused by the fact that the quality of the lower bound is higher for bigger cycle times. Exchanging our simple lower bound by a more sophisticated lower bound might help to improve the performance of Beam-ACO when smaller cycle times are concerned.

[3] This was done by setting $a_j := t_{n-j+1}$ for all $j \in T$, and $a := c$.

4.3 Results for the Nissan TSALBP-1 instance

Finally we applied Beam-ACO to the real-life instance provided by the Nissan plant in Barcelona, Spain. This instance consists of 140 tasks, a cycle time of 180 seconds, and a space limit of 4. This real-life instances is easily solvable: Our Beam-ACO approach finds in a fraction of a second an optimal solution with 21 work stations. Also when disregarding space constraints (that is, regarding it as a SALBP-1 isntance), it is easily solvable. The optimal solution in this case has 17 work stations.

5 Conclusions and outlook to the future

In this work we have proposed a hybrid ant colony optimization approach called Beam-ACO for the TSALBP-1 problem. Beam-ACO is obtained by hybridizing ant colony optimization with beam search. Our approach differs from existing ant colony optimization approaches for TSALBP-1 (and from most existing metaheuristics) by the fact that it solves the problem in a direct way. The results show that Beam-ACO is a state-of-the-art metaheuristics, for example, for the well-studied SALBP-1 problem, which is a specific case of TSALBP-1.

In the future we plan to study the influence of the different algorithmic components on the performance of Beam-ACO. It would be interesting to know, for example, which influence the heuristic information (obtained by parametrizing the priority rule) exactly has. Furthermore, we plan to extend the experimental evaluation of the algorithm to beam widths greater than one. We expect that this will further improve the algorithms' performance.

References

[1] K. R. Baker. *Introduction to sequencing and scheduling.* Wiley, New York, 1974.

[2] J. Bautista and J. Pereira. Ant algorithms for assembly line balancing. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms – Proceedings of ANTS 2002 – Third International Workshop*, volume 2463 of *Lecture Notes in Computer Science*, pages 65–75. Springer Verlag, Berlin, Germany, 2002.

[3] J. Bautista and J. Pereira. Ant algorithms for a time and space constrained assembly line balancing problem. *European Journal of Operational Research*, 2006. In press.

[4] C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

[5] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.

[6] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, 2004.

[7] A. Fügenschuh. Parametrized greedy heuristics in theory and practice. In M. J. Blesa, C. Blum, A. Roli, and M. Sampels, editors, *Proceedings of HM 2005 – 2nd International Workshop on Hybrid Metaheuristics*, volume 3636 of *Lecture Notes in Computer Science*, pages 21–31. Springer Verlag, Berlin, Germany, 2005.

[8] J. F. Gonçalves and J. R. de Almeida. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8:629–642, 2002.

[9] T. R. Hoffmann. EUREKA: A hybrid system for assembly line balancing. *Management Science*, 38:39–47, 1992.

[10] R. V. Johnson. Optimally balancing large assembly lines with "FABLE". *Management Science*, 34:240–253, 1988.

[11] D. L. Lapierre, A. Ruiz, and P. Soriano. Balancing assembly lines with tabu search. *European Journal of Operational Research*, 168:826–837, 2006.

[12] D. Merkle and M. Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In *Proceedings of the EvoWorkshops 2000*, volume 1803 of *Lecture Notes in Computer Science*, pages 287–296. Springer Verlag, Berlin, Germany, 2000.

[13] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.

[14] A. Scholl and C. Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, 2006.

[15] A. Scholl and R. Klein. SALOME: A bidirectional branch and bound procedure for assembly line balancing. *INFORMS Journal on Computing*, 9:319–334, 1997.

[16] A. Scholl and S. Voss. Simple assembly line balancing—Heuristic approaches. *Journal of Heuristics*, 2:217–244, 1996.

[17] Y. H. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *In the proceedings of the IEEE International Conference on Evolutionary Computation*, pages 96–73. IEEE press, 1998.

[18] K. Socha. ACO for continuous and mixed-variable optimization. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 25–36. Springer Verlag, Berlin, Germany, 2004.

[19] A. Sprecher. Dynamic search tree decomposition for balancing assembly lines by parallel search. *International Journal of Production Research*, 41:1423–1430, 2003.

[20] F. B. Talbot, J. H. Patterson, and J. H. Gehrlein. A comparative evaluation of heuristic line balancing techniques. *Management Science*, 32:430–454, 1986.

[21] P. M. Vilarinho and A. Simaria. A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *International Journal of Production Research*, pages 1405–1420, 2002.

Table 3: Results obtained by Beam-ACO in comparison to the results of two of the best techniques available for SALBP-1: ANTS is a standard ACO approach proposed in [3], and TABU is a recent tabu search approach proposed in [11]. The comparison is performed on the 26 difficult instances based on the precedence graph called SCHOLL. The instances differ in the cycle time, which is indicated in the first table column. The second column (headed by **bks**) contains the best known solution, and the third and fourth column contain the values of the best solutions found by ANTS, respectively TABU. Finally, the last 3 table columns provide the results of Beam-ACO, concerning the best solution found in 10 runs (**best**), the average and standard deviation of the results (**average (std)**), and the times including the standard deviation at which the best solutions were found (**average time (std)**).

c	**bks**	**ANTS**	**TABU**	**Beam-ACO**		
				best	**average (std)**	**average time (std)**
1394	50	52	51	51	51.00 (0.00)	0.82 (0.87)
1422	50	51	**50**	**50**	50.00 (0.00)	0.29 (0.39)
1452	48	50	49	49	49.00 (0.00)	0.78 (0.99)
1483	47	49	48	48	48.00 (0.00)	1.47 (1.35)
1515	46	48	47	47	47.00 (0.00)	0.39 (0.77)
1548	46	**46**	**46**	**46**	46.00 (0.00)	0.67 (0.26)
1584	44	46	45	45	45.00 (0.00)	1.48 (0.92)
1620	44	**44**	**44**	**44**	44.00 (0.00)	1.27 (0.82)
1659	42	44	43	43	43.00 (0.00)	0.52 (0.57)
1699	42	**42**	**42**	**42**	42.00 (0.00)	2.71 (0.29)
1742	40	41	41	41	41.00 (0.00)	0.47 (0.33)
1787	39	40	40	40	40.00 (0.00)	0.59 (0.27)
1834	38	39	39	39	39.00 (0.00)	0.34 (0.27)
1883	37	38	38	38	38.00 (0.00)	0.022 (0.024)
1935	36	37	37	37	37.00 (0.00)	0.21 (0.15)
1991	35	37	36	**35**	35.90 (0.32)	0.33 (0.97)
2049	34	35	35	35	35.00 (0.00)	0.013 (0.0057)
2111	33	34	34	34	34.00 (0.00)	0.010 (0.0042)
2177	32	33	33	**32**	32.90 (0.32)	9.28 (29.31)
2247	31	32	32	32	32.00 (0.00)	0.0090 (0.0054)
2322	30	31	31	31	31.00 (0.00)	0.012 (0.0090)
2402	29	30	30	30	30.00 (0.00)	0.010 (0.0051)
2488	28	29	29	29	29.00 (0.00)	0.011 (0.0047)
2580	27	28	28	**27**	27.80 (0.42)	14.02 (31.99)
2680	26	27	27	**26**	26.10 (0.32)	47.14 (36.68)
2787	25	26	26	**25**	25.20 (0.42)	50.19 (30.97)

Table 4: Results obtained by Beam-ACO in comparison to the results of ANTS [3], which is so far the only available technique for TSALBP-1. The comparison is performed on the 26 instances based on the precedence graph called SCHOLL. The instances differ in the cycle time (which is at the same time the space limit). Cycle time, respectively space limit, are indicated in the first table column. The second column (headed by **bks**) contains the values of the best known solution. The arrow indicates that Beam-ACO was the first algorithm to generate this best known solution value. The third column contains the values of the best solutions found by ANTS. Finally, the last 3 table columns provide the results of Beam-ACO, concerning the best solution found in 10 runs (**best**), the average and standard deviation of the results (**average (std)**), and the times including the standard deviation at which the best solutions were found(**average time (std)**).

c, a	**bks**	**ANTS**	**Beam-ACO**		
			best	**average (std)**	**average time (std)**
1394	→ 59	60	**59**	60.00 (0.47)	30.50 (35.27)
1422	58	**58**	59	59.00 (0.00)	9.21 (10.07)
1452	→ 57	58	**57**	57.40 (0.52)	13.82 (36.23)
1483	→ 55	56	**55**	56.20 (0.63)	34.23 (33.35)
1515	54	**54**	**54**	54.40 (0.52)	51.43 (40.23)
1548	53	**53**	**53**	53.10 (0.32)	3.22 (5.79)
1584	→ 51	53	**51**	51.70 (0.48)	26.61 (38.82)
1620	→ 49	50	**49**	49.70 (0.48)	15.79 (23.57)
1659	→ 48	49	**48**	48.30 (0.48)	40.80 (34.67)
1699	→ 46	47	**46**	46.90 (0.32)	14.33 (14.77)
1742	→ 45	46	**45**	45.00 (0.00)	38.65 (31.60)
1787	→ 44	45	**44**	44.00 (0.00)	20.90 (21.57)
1834	43	**43**	**43**	43.00 (0.00)	13.13 (7.59)
1883	42	**42**	**42**	42.00 (0.00)	9.04 (4.05)
1935	41	**41**	**41**	41.00 (0.00)	9.36 (5.87)
1991	40	**40**	**40**	40.00 (0.00)	6.24 (3.45)
2049	→ 38	39	**38**	38.00 (0.00)	12.64 (10.05)
2111	37	**37**	**37**	37.00 (0.00)	2.02 (4.28)
2177	36	**36**	**36**	36.00 (0.00)	2.06 (3.70)
2247	→ 34	35	**34**	34.80 (0.42)	10.37 (19.81)
2322	→ 33	34	**33**	33.40 (0.52)	33.66 (36.84)
2402	→ 32	33	**32**	32.70 (0.48)	27.83 (45.99)
2488	→ 31	32	**31**	31.00 (0.00)	42.11 (22.34)
2580	30	**30**	**30**	30.00 (0.00)	4.20 (3.00)
2680	29	**29**	**29**	29.00 (0.00)	3.83 (3.92)
2787	28	**28**	**28**	28.00 (0.00)	4.41 (4.15)