ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Operations Research ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

computers & operations research

www.elsevier.com/locate/dsw

Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling

Christian Blum*

IRIDIA, Université Libre de Bruxelles, CP 194/6, Av. Franklin D. Roosevelt 50, Bruxelles 1050, Belgium

Abstract

Ant colony optimization (ACO) is a metaheuristic approach to tackle hard combinatorial optimization problems. The basic component of ACO is a probabilistic solution construction mechanism. Due to its constructive nature, ACO can be regarded as a tree search method. Based on this observation, we hybridize the solution construction mechanism of ACO with beam search, which is a well-known tree search method. We call this approach *Beam-ACO*. The usefulness of Beam-ACO is demonstrated by its application to open shop scheduling (OSS). We experimentally show that Beam-ACO is a state-of-the-art method for OSS by comparing the obtained results to the best available methods on a wide range of benchmark instances.

Keywords: Ant colony optimization; Beam search; Tree search; Open shop scheduling

1. Introduction

Among the approximate methods for solving combinatorial optimization (CO) problems [1] we can identify two large groups: tree search methods [2] and local search methods [3]. The nature of tree search methods is constructive. The solution construction mechanism maps the search space to a tree structure, where a path from the root node to a leaf corresponds to the process of constructing a solution. Then, the search space is explored by repeated or parallel solution constructions. In contrast, local search methods explore a search space by moving from solution to solution on a landscape that is imposed by a neighborhood structure on the search space. The simplest example is a steepest descent local search that moves at each step from the current solution to the best neighbor of the current solution.

* Fax: +32-2-650-2715.
E-mail address: cblum@ulb.ac.be (C. Blum).

doi:10.1016/j.cor.2003.11.018

Most of the classical tree search methods have their origin in the fields of operations research (OR) or artificial intelligence (AI). Examples are greedy heuristics [1], backtracking methods [2], and beam search (BS) [4]. They are often relaxations or derivations of exact methods such as branch and bound [1]. In the past 15–20 years, metaheuristics [5,6] emerged as alternative approximate methods for solving CO problems. Most of the metaheuristic techniques are based on local search. Examples are tabu search (TS) [7], simulated annealing (SA) [8], and iterated local search (ILS) [9]. However, other metaheuristics such as the greedy randomized adaptive search procedure (GRASP) [10] can be regarded as probabilistic tree search methods (see [11]).

1.1. Our contribution

An interesting example of a metaheuristic that can be seen as a probabilistic tree search method is ant colony optimization (ACO) [12,13]. In ACO algorithms, artificial ants construct solutions from scratch by probabilistically making a sequence of local decisions. At each construction step an ant chooses exactly one of possibly several ways of extending the current partial solution. The rules that define the solution construction mechanism in ACO implicitly map the search space of the considered problem (including the partial solutions) onto a search tree. This view of ACO as a tree search procedure allows us to put ACO into relation with classical tree search methods such as beam search (BS) [14]. One of the interesting features of BS is that it works on a set of partial solutions in parallel, extending each partial solution—in contrast to ACO—at each step in several possible ways. However, in BS the extension of partial solutions is usually done by using a deterministic greedy policy with respect to a weighting function that gives weights to the possible extensions. The idea of this paper is to hybridize the solution construction mechanism of ACO with BS, which results in a general approach that we call Beam-ACO. We apply Beam-ACO to open shop scheduling (OSS) [4]. We show that Beam-ACO improves on the results obtained by the best standard ACO approach for OSS that was proposed in [5]. Furthermore, we show that Beam-ACO is a state-of-the-art method for the OSS problem by comparing it to the genetic algorithm by Liaw [16] and to the genetic algorithm by Prins [17].

1.2. Related work

The connection between ACO and tree search techniques was established before in [18–20]. For example in [18], the author describes an ACO algorithm for the quadratic assignment problem (QAP) as an approximate non-deterministic tree search procedure. The results of this approach are compared to both exact algorithms and BS techniques. Recently, an ACO approach to set partitioning (SP) that allowed the extension of partial solutions in several possible ways was presented in [19]. Furthermore, ACO has been described from a dynamic programming (DP) [21] perspective in [20], where ants are described as moving on a tree structure.

The outline of the paper is as follows. In Section 2 we explain the concept of a search tree. In Section 3 we briefly outline ACO and BS, before we outline the general concepts of Beam-ACO in Section 4. In Section 5 we introduce the OSS problem and propose a Beam-ACO algorithm to tackle this problem. Finally, in Section 6 we provide an experimental evaluation of Beam-ACO and we offer a summary and an outlook to the future in Section 7.

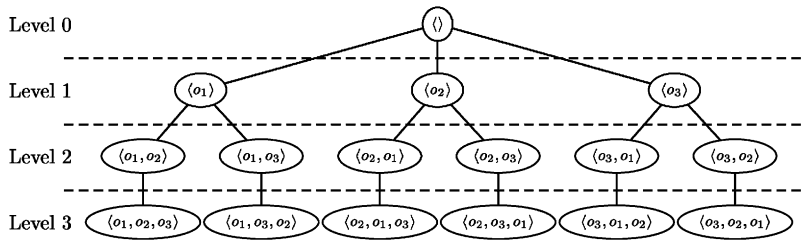


Fig. 1. The search tree for an OSS instance on three operations that is defined by the construction mechanism of building permutations of all operations from left to right. The inner nodes of the search tree are partial solutions, whereas the leaves are solutions.

2. Combinatorial optimization and search trees

According to [1], a CO problem $\mathcal{P} = (\mathcal{S}, f)$ is an optimization problem in which is given a finite set of objects $\mathcal{S}$ and an objective function $f : \mathcal{S} \mapsto \mathbb{R}^+$ that assigns a positive cost value to each of the objects. The goal is to find an object of minimal cost value.[1] The objects are solutions to the problem.

The solutions of a CO problem are generally composed of solution components from a finite set of solution components. As an example consider the OSS problem in which the set of solution components consists of n operations $\mathcal{O} = \{o_1, \dots, o_n\}$. Each permutation of the n operations is a solution to the problem instance under consideration.

Constructive methods are characterized by the fact that they work on partial solutions. Each constructive method is based on a mechanism for extending partial solutions. Generally, a partial solution s^p is extended by adding a solution component from the set $\mathcal{N}(s^p)$ of solution components that are allowed to be added. This set is generally defined by the problem constraints. A mechanism for extending partial solutions defines a *search tree* in the following way. The root of a search tree is the empty partial solution $s^p = \langle\rangle$, which is the first level of the search tree. The second level of the search tree consists of all partial solutions that can be generated by adding a solution component to the empty partial solution, and so on. Finally, the leaves of a search tree consist either of solutions to the problem instance under consideration, or possibly partial solutions that cannot be further extended. As an example we consider a construction mechanism for the OSS problem that builds permutations of all operations from left to right by extending a partial solution s^p at each step through adding one of the solution components from $\mathcal{N}(s^p) = \{o_1, \dots, o_n\} \setminus \{o_i \mid o_i \in s^p\}$. The definition of $\mathcal{N}(s^p)$ results from the fact that only a permutation of all the operations corresponds to a solution, which implies that an operation o_i can be added exactly once. The corresponding search tree for an OSS problem instance on three operations (i.e., $n = 3$) is shown in Fig. 1.

3. Ant colony optimization and beam search

The simplest method that operates on a search tree is a greedy heuristic which builds solutions to a CO problem as follows. Starting from the empty partial solution $s^p = \langle\rangle$, the current partial solution

---

[1] Note that minimizing over an objective function f is the same as maximizing over $-f$. Therefore, every CO problem can be described as a minimization problem.

s^p is extended at each step by adding a solution component from the set $\mathcal{N}(s^p)$. This is done until the current partial solution can either not be further extended or a complete solution is constructed. A weighting function $\eta : \mathcal{N}(s^p) \mapsto \mathbb{R}^+$ is used to assign a weight to each possible extension of the current partial solution. These weights reflect the benefit of extending a partial solution in different ways with respect to a certain measure. According to the greedy policy, at each construction step one of the extensions with the highest weight is chosen.[2] A greedy heuristic follows exactly one path from the root node to one of the leaf nodes of the search tree.

The main drawback of a greedy heuristic is quite obvious. It depends strongly on the quality of the weighting function that is used. In the following we briefly describe ACO and BS, which try to overcome this dependency on the weighting function in very different ways.

3.1. Ant colony optimization

Ant colony optimization (ACO) [12,13] is a metaheuristic to tackle hard CO problems that was first proposed in the early 1990s [22–24]. The inspiring source of ACO is the foraging behaviour of real ants. ACO algorithms are characterized by the use of a (parametrized) probabilistic model that is used to probabilistically construct solutions to the problem under consideration. The probabilistic model is called the *pheromone model*. The pheromone model consists of a set of model parameters $\mathcal{T}$, that are called the *pheromone trail parameters*. The pheromone trail parameters $\mathcal{T}_i \in \mathcal{T}$ have values τ_i, called *pheromone values*. Usually, pheromone trail parameters are associated to the solution components (or subsets of solution components). ACO algorithms are iterative processes that are terminated by stopping conditions such as a maximum CPU time. At each iteration, n_a ants probabilistically construct solutions to the problem under consideration. Then, optionally a local search procedure is applied to improve the constructed solutions. Finally, some of the constructed solutions are used for performing an update of the pheromone values. The pheromone update aims at increasing the probability to generate high quality solutions over time.

Algorithm 1. Solution construction in ACO.
input: the empty partial solution $s^p = \langle\rangle$.
while $\mathcal{N}(s^p) \neq \emptyset$ **do**
  Choose $c \in \mathcal{N}(s^p)$ according to probability $\mathbf{p}(c \mid \mathcal{T}, \eta)$ {see text}
  $s^p \leftarrow$ extend s^p by adding solution component c
end while
output: a solution s (resp., partial solution s^p, in case s^p is partial and can not be extended)

The solution construction process in ACO algorithms (see Algorithm 1) is equivalent to a greedy heuristic, except that the choice of the next solution component at each step is done probabilistically instead of deterministically. The probabilities are called *transition probabilities*, henceforth denoted by $\mathbf{p}(c \mid \mathcal{T}, \eta)$, $\forall c \in \mathcal{N}(s^p)$. They are a function of the pheromone values and a weighting function η. The weights assigned by a weighting function are in the context of ACO algorithms commonly called the heuristic information. An interesting feature is that the pheromone value update makes the search process that is performed by ACO algorithms adaptive in the sense that the accumulated search experience is used in order to direct the future search process.

---

[2] An example of a greedy policy is the well-known nearest-neighbor policy for constructing solutions to the travelling salesman problem (TSP) [1].

3.2. Beam search

BS is a classical tree search method that was introduced in the context of scheduling [14], but has since then been successfully applied to many other CO problems. BS algorithms are incomplete derivatives of branch and bound algorithms, and are therefore approximate methods.

The central idea behind BS is to allow the extension of partial solutions in several possible ways. At each step the algorithm extends each partial solution from a set $\mathscr{B}$, which is called the *beam*, in at most k_{ext} possible ways. Each newly obtained partial solution is either stored in the set of complete solutions $\mathscr{B}_{\text{c}}$ (in case it is a complete solution), or in the set $\mathscr{B}_{\text{ext}}$ (in case it is a further extensible partial solution). At the end of each step, the algorithm creates a new beam $\mathscr{B}$ by selecting up to k_{bw} (called the *beam width*) solutions from the set of further extensible partial solutions $\mathscr{B}_{\text{ext}}$. In order to select partial solutions from $\mathscr{B}_{\text{ext}}$, BS algorithms use a mechanism to evaluate partial solutions. An example of such a mechanism is a lower bound. Given a partial solution s^p, a lower bound computes the minimum objective function value for any complete solution s that can be constructed starting from s^p.

At the first step, the beam $\mathscr{B}$ only consists of the empty partial solution $s^p = \langle\rangle$. As for greedy heuristics, the extension of partial solutions is done by applying a deterministic greedy policy based on a weighting function η. The algorithmic framework of BS is shown in Algorithm 2. In this framework, the procedure PreSelect($\mathscr{N}(s^p)$) is optional and is, in some applications, used to filter the set of possible extensions of a partial solution.

Algorithm 2. Beam search (BS)
input: an empty partial solution $s^p = \langle\rangle$, beam width k_{bw}, max. number of extensions k_{ext}
$\mathscr{B} \leftarrow \{s^p\}$, $\mathscr{B}_{\text{c}} \leftarrow \emptyset$
while $\mathscr{B} \neq \emptyset$ **do**
  $\mathscr{B}_{\text{ext}} \leftarrow \emptyset$
  for each $s^p \in \mathscr{B}$ **do**
    $count \leftarrow 1$
    $\mathscr{N}(s^p) \leftarrow$ PreSelect($\mathscr{N}(s^p)$) {optional}
    while $count \leqslant k_{\text{ext}}$ **AND** $\mathscr{N}(s^p) \neq \emptyset$ **do**
      Choose $c \leftarrow \text{argmax}\{\eta(c) \mid c \in \mathscr{N}(s^p)\}$
      $s^{p\prime} \leftarrow$ extend s^p by adding solution component c
      $\mathscr{N}(s^p) \leftarrow \mathscr{N}(s^p) \setminus \{c\}$
      if $s^{p\prime}$ extensible **then**
        $\mathscr{B}_{\text{ext}} \leftarrow \mathscr{B}_{\text{ext}} \cup \{s^{p\prime}\}$
      else
        $\mathscr{B}_{\text{c}} \leftarrow \mathscr{B}_{\text{c}} \cup \{s^{p\prime}\}$
      end if
      $count \leftarrow count + 1$
    end while
  end for
  Rank the partial solutions in $\mathscr{B}_{\text{ext}}$ using a lower bound $LB(\cdot)$
  $\mathscr{B} \leftarrow$ select the $\min\{k_{\text{bw}}, |\mathscr{B}_{\text{ext}}|\}$ highest ranked partial solutions from $\mathscr{B}_{\text{ext}}$
end while
output: a set of candidate solutions $\mathscr{B}_{\text{c}}$

The existence of an accurate—and computationally inexpensive—lower bound is crucial for the success of BS.[3] To summarize, a BS technique constructs several candidate solutions in parallel and uses a lower bound in order to guide the search. BS methods are usually deterministic (i.e. the policy that is used for extending partial solutions is deterministic).

4. Beam-ACO

As we have outlined in the previous section, ACO and BS have the common feature that they are both based on the idea of constructing candidate solutions step-by-step. However, the ways by which the two methods explore the search space are quite different. BS algorithms are guided by two different components. These are (i) the weighting function that is used to weight the different possibilities of extending a partial solution, and (ii) the lower bound that is used for restricting the number of partial solutions at each step. As mentioned before, the policy that is used in BS algorithms for extending partial solutions is usually deterministic. In contrast, ACO algorithms explore the search space in a probabilistic way, using past search experience in order to find good areas of the search space. In other words, ACO's search process is adaptive.

In general, BS algorithms have the advantage that they are strongly guided by the deterministic use of a weighting function and by the heuristic guidance of the lower bound. However, the use of a deterministic policy for extending partial solutions may reduce, in case the weighting function is bad, the chances of finding high-quality solutions. On the other side, in ACO algorithms randomization often helps in searching around presumably good solutions. However, the method is highly sensible to the balance between heuristic guidance and randomization.

Based on these considerations we expect a benefit from combining these two ways of exploring a search space. The basic algorithmic framework of our new approach is the framework of ACO. However, we replace the solution construction mechanism of standard ACO algorithms by a solution construction mechanism in which each artificial ant performs a probabilistic BS. This probabilistic BS is obtained from Algorithm 2 by replacing the deterministic choice of a solution component at each construction step by a probabilistic choice based on transition probabilities. As the transition probabilities depend on the changing pheromone values, the probabilistic beam searches that are performed by this algorithm are also adaptive. We call this new approach *Beam-ACO*.

There are basically three design choices to be made when developing a Beam-ACO approach. The first one concerns the lower bound $LB(\cdot)$ that is used to evaluate partial solutions. If no accurate lower bound that can be computed in an efficient way can be found, the Beam-ACO approach might fail. The second design decision concerns the setting of the parameters k_{bw} and k_{ext}. Both parameter values may be static or dynamically changing, depending on the state of the solution construction. For example, it might be beneficial to allow more extensions of partial solutions in early stages of the construction process than in later stages. Finally, the third design decision concerns the possibility that the set of solution components $\mathcal{N}(s^p)$ that can be used to extend a partial solution s^p might be

[3] An inaccurate lower bound might bias the search towards bad areas in the search space.

effectively restricted by a pre-selection mechanism. Such a pre-selection mechanism is performed in procedure PreSelect($\mathcal{N}(s^p)$) of Algorithm 2.

5. Application: open shop scheduling

In order to show the usefulness of our idea we developed a Beam-ACO approach, henceforth denoted by Beam-ACO-OSS, for the OSS problem [4]. In the field of scheduling, ACO has so far been successfully applied to the single machine weighted tardiness (SMWT) problem [25], and to the resource constraint project scheduling (RCPS) problem [26]. However, the application to shop scheduling problems has proved to be quite difficult. The earliest ACO algorithm to tackle a shop scheduling problem was the one by Colorni et al. [27] for the job shop scheduling problem. The results obtained by this algorithm are quite far from the state-of-the-art. The first quite successful ACO algorithm to tackle shop scheduling problems was proposed in [15]. This algorithm was developed for the group shop scheduling (GSS) problem [28], which is a very general shop scheduling problem that includes the flow shop (FSS), the job shop (JSS), and the open shop scheduling (OSS) problem. Despite its generality, this approach achieves—especially for OSS problem instances—good results. In the following we first outline Beam-ACO-OSS, before we show (1) that Beam-ACO-OSS improves on the results of the currently best standard ACO algorithm [15] (henceforth denoted by Standard-ACO-OSS), and (2) that Beam-ACO-OSS is a new state-of-the-art algorithm for solving the existing OSS benchmark instances.

5.1. Open shop scheduling

The OSS problem can be formalized as follows. We consider a finite set of operations $\mathcal{O} = \{o_1, \ldots, o_n\}$ which is partitioned into subsets $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_{|\mathcal{M}|}\}$. The operations in $\mathcal{M}_i \in \mathcal{M}$ have to be processed on the same machine. For the sake of simplicity we identify each set $\mathcal{M}_i \in \mathcal{M}$ of operations with the machine they have to be processed on, and call $\mathcal{M}_i$ a machine. Set $\mathcal{O}$ is additionally partitioned into subsets $\mathcal{J} = \{\mathcal{J}_1, \ldots, \mathcal{J}_{|\mathcal{J}|}\}$, where the set of operations $\mathcal{J}_j \in \mathcal{J}$ is called a job. Furthermore, given is a function $p : \mathcal{O} \to \mathbb{N}^+$ that assigns processing times to operations. We consider the case in which each machine can process at most one operation at a time. Operations must be processed without preemption (that is, once the processing of an operation has started it must be completed without interruption). Operations belonging to the same job must be processed sequentially.

A solution is given by permutations $\pi^{\mathcal{M}_i}$ of the operations in $\mathcal{M}_i$, $\forall i \in \{1, \ldots, |\mathcal{M}|\}$, and permutations $\pi^{\mathcal{J}_j}$ of the operations in $\mathcal{J}_j$, $\forall j \in \{1, \ldots, |\mathcal{J}|\}$. These permutations define processing orders on all the subsets $\mathcal{M}_i$ and $\mathcal{J}_j$. Note that not all combinations of permutations are feasible, because some combinations of permutations might define cycles in the processing orders. As mentioned in Section 2, a permutation of all the operations represents a solution to an OSS instance. This is because a permutation of all operations contains the permutations of the operations of each job and of each machine. In the following we refer to the search space $\mathcal{S}$ as the set of all permutations of all operations.

There are several possibilities to measure the cost of a solution. In this paper we deal with makespan minimization. Every operation $o \in \mathcal{O}$ has a well-defined *earliest starting time* $t_{\mathrm{es}}(o, s)$ with

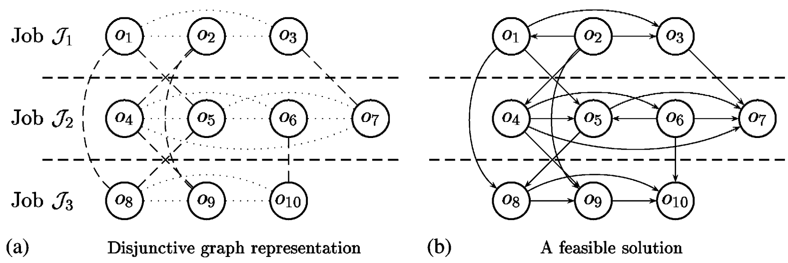


Fig. 2. (a) The disjunctive graph representation [29] of a simple instance of the OSS problem consisting of 10 operations partitioned into 3 jobs, and 4 machines (processing times are omitted in this example). Problem specification: $\mathcal{O} = \{o_1, \ldots, o_{10}\}$, $\mathcal{J} = \{\mathcal{J}_1 = \{o_1, o_2, o_3\}, \mathcal{J}_2 = \{o_4, \ldots, o_7\}, \mathcal{J}_3 = \{o_8, o_9, o_{10}\}\}$, $\mathcal{M} = \{\mathcal{M}_1 = \{o_1, o_5, o_8\}, \mathcal{M}_2 = \{o_2, o_4, o_9\}, \mathcal{M}_3 = \{o_3, o_7\}, \mathcal{M}_4 = \{o_6, o_{10}\}\}$. The nodes of the graph correspond to the operations. Furthermore, there are undirected arcs between every pair of operations being in the same job (dotted) or having to be processed on the same machine (dashed). In order to obtain a solution, the undirected arcs have to be directed without creating any cycles. This corresponds to finding permutations of the operations on the machines and in the jobs such that there are no cycles in the processing orders. (b) A feasible solution to the problem. The undirected arcs from (a) are directed and the new directed graph does not contain any cycles.

respect to a solution s (respectively, a well-defined earliest starting time $t_{\text{es}}(o, s^p)$ with respect to a partial solution s^p). Here we assume that all the operations that do not have any predecessor have an earliest starting time of 0. Accordingly, the *earliest completion time* of an operation $o \in \mathcal{O}$ with respect to a solution s is denoted by $t_{\text{ec}}(o, s)$ and defined as $t_{\text{es}}(o, s) + p(o)$.[4] The same definitions hold for partial solutions. The objective function value $f(s)$ of a feasible solution s (also called the makespan of s) is given by the maximum of the earliest completion times of all the operations:

$$f(s) \leftarrow \max\{t_{\text{ec}}(o, s) \mid o \in \mathcal{O}\}. \tag{1}$$

We aim at minimizing f. An example of an OSS instance is shown as a disjunctive graph in Fig. 2.

5.2. Beam-ACO-OSS

The algorithmic framework of Beam-ACO-OSS is shown in Algorithm 3. This basic ACO framework works as follows: At each iteration, n_a ants perform a probabilistic beam search. Hereby, each ant constructs up to k_{bw} solutions. Then, depending on a measure that is called the convergence factor and a boolean control variable *bs_update*, an update of the pheromone values is performed. After that, the convergence factor is recomputed and it is decided if the algorithm has to be restarted. The algorithm stops when the termination conditions are satisfied. One of the most important ingredients of an ACO algorithm is the pheromone model. The one that we used is defined in the following. Note that this pheromone model was also used for Standard-ACO-OSS in [15].

[4] Remember that $p(o)$ is the processing time of an operation o.

Algorithm 3. Beam-ACO for open shop scheduling (Beam-ACO-OSS)
input: an OSS problem instance
$s_{bs} \leftarrow$ NULL, $s_{rb} \leftarrow$ NULL, $cf \leftarrow 0$, $bs\_update \leftarrow FALSE$
InitializePheromoneValues($\mathcal{T}$)
while termination conditions not satisfied **do**
$\mathcal{S}_{iter} \leftarrow \emptyset$
for $j \leftarrow 1$ to n_a
$\mathcal{S}_{iter} \leftarrow \mathcal{S}_{iter} \cup$ BeamACOSolutionConstruction($\mathcal{T}$) {See Section 5.2.1, Algorithm 4}
end for
ApplyLocalSearch($\mathcal{S}_{iter}$)
$s_{ib} \leftarrow \text{argmin}\{f(s) \mid s \in \mathcal{S}_{iter}\}$
if s_{rb}= NULL or $f(s_{ib}) < f(s_{rb})$ **then** $s_{rb} \leftarrow s_{ib}$
if s_{bs}= NULL or $f(s_{ib}) < f(s_{bs})$ **then** $s_{bs} \leftarrow s_{ib}$
ApplyPheromoneUpdate(bs_update,$\mathcal{T}$,s_{rb},s_{bs})
$cf \leftarrow$ ComputeConvergenceFactor()
if $cf > cf\_limit$ **then**
if $bs\_update = TRUE$ **then**
ResetPheromoneValues($\mathcal{T}$)
$s_{rb} \leftarrow$ NULL
$bs\_update \leftarrow FALSE$
else
$bs\_update \leftarrow TRUE$
end if
end if
end while
output: s_{bs}, the best solution found

Definition 1. Two operations $o_i, o_j \in \mathcal{O}$ are called related, if they are either in the same job, or if they have to be processed on the same machine. The set of operations that is related to an operation o_i is in the following denoted by $\mathcal{R}_i$. Then, the pheromone model consists for each pair of related operations $o_i, o_j \in \mathcal{O}$ of a pheromone trail parameter $\mathcal{T}_{ij}$ and a pheromone trail parameter $\mathcal{T}_{ji}$. The value τ_{ij} of pheromone trail parameter $\mathcal{T}_{ij}$ encodes the desirability of processing o_i before o_j, whereas the value τ_{ji} of pheromone trail parameter $\mathcal{T}_{ji}$ encodes the desirability of processing o_j before o_i.

The components of Beam-ACO-OSS are outlined in more detail in the following.

InitializePheromoneValues($\mathcal{T}$): At the start of the algorithm all pheromone values are initialized to 0.5.[5]

ApplyLocalSearch($\mathcal{S}_{iter}$): To every solution $s \in \mathcal{S}_{iter}$, where $\mathcal{S}_{iter}$ is the set of solutions that was constructed by the ants at the current iteration, we apply a steepest descent local search procedure

[5] This is reasonable as our algorithm is implemented in the hyper-cube framework [30,31], which limits the pheromone values between 0 and 1.

that is based on the neighborhood structure proposed in [28] for the group shop scheduling problem. [6]

ApplyPheromoneUpdate($bs\_update, \mathcal{T}, s_{rb}, s_{bs}$): For updating the pheromone values at each iteration we either use the restart best solution s_{rb} or the best-so-far solution s_{bs}. The pheromone update rule is as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \rho(\delta(o_i, o_j, s) - \tau_{ij}), \quad \forall \mathcal{T}_{ij} \in \mathcal{T}, \tag{2}$$

where

$$\delta(o_i, o_j, s) = \begin{cases} 1 & \text{if } o_i \text{ is scheduled before } o_j \text{ in } s, \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

and $\rho \in (0, 1]$ is a constant called evaporation rate. The solution that is used for updating the pheromone values depends on the status of the boolean variable *bs_update*. At the (re-)start of the algorithm it holds that $bs\_update = FALSE$, and we set $s \leftarrow s_{rb}$. Once the algorithm has converged (which is indicated by $cf > cf\_limit$), the setting changes to $bs\_update = TRUE$, and we set $s \leftarrow s_{bs}$ until the algorithm has converged again. Then, the algorithm is restarted by resetting all the pheromone values to 0.5.

Furthermore, we applied an upper bound τ_{max} and a lower bound τ_{min} to the pheromone values as proposed in [32] for $\mathcal{MAX}$–$\mathcal{MIN}$ ant systems ($\mathcal{MM}$ASs). This prevents the algorithm from converging to a solution. [7] For all our experiments we have set the lower bound τ_{min} to 0.001 and the upper bound τ_{max} to 0.999, as well as *cf_limit* to 0.99. Therefore, after applying the pheromone update rule above, we check which pheromone values exceed the upper bound, or are below the lower bound. These pheromone values are then set back to the respective bound.

ComputeConvergenceFactor(): As mentioned above, our ACO algorithm is controlled by a numerical factor that is called the *convergence factor*, and that is denoted by $cf \in [0, 1]$. This factor is computed as follows:

$$cf \leftarrow 2 \cdot \left(\left(\frac{\sum_{o_i \in \mathcal{O}} \sum_{o_j \in \mathcal{R}_i} \max\{\tau_{max} - \tau_{ij}, \tau_{ij} - \tau_{min}\}}{|\mathcal{T}|(\tau_{max} - \tau_{min})} \right) - 0.5 \right). \tag{4}$$

Therefore, when the algorithm is initialized (or reset) with all pheromone values set to 0.5, then $cf = 0$, while when the algorithm has converged, then $cf = 1$.

ResetPheromoneValues($\mathcal{T}$): This function sets all the pheromone values back to the constant 0.5.

This concludes the description of the algorithmic framework. In the following we outline the solution construction mechanism (i.e., function BeamACOSolutionConstruction($\mathcal{T}$)) of Beam-ACO-OSS.

5.2.1. *Solution construction in* Beam-ACO-OSS

For constructing solutions we use the mechanism of the list scheduler algorithm [29], which is a widely used algorithm for constructing feasible solutions to shop scheduling problems. The list scheduler algorithm starts from the empty partial solution and builds a permutation of all the operations from left to right by appending at each construction step another operation to the current

---

[6] OSS is a special case of the more general group shop scheduling problem.

[7] We say that an ACO algorithm has converged to a solution s if only s has a probability greater than ε to be generated (with ε close to zero).

partial solution (henceforth called scheduling an operation). At each construction step t the current partial solution s_t^p induces a partition of the set of operations $\mathcal{O}$ into the set of operations $\mathcal{O}_t^- = \{o_i \mid o_i \in s_t^p\}$ and the set of operations $\mathcal{O}_t^+ = \mathcal{O} \setminus \mathcal{O}_t^-$. In principle, all the operations in $\mathcal{O}_t^+$ can be used to extend the partial solution s_t^p. However, if for an operation $o_i \in \mathcal{O}_t^+$ it holds that $\mathcal{R}_i \cap \mathcal{O}_t^+ = \emptyset$,[8] we do not need to consider it as a candidate for extending the current partial solution, because as no related operations are left, the position of this operation in the final permutation is meaningless.[9] Therefore, at each construction step t the set of allowed operations is defined as follows:

$$\mathcal{N}(s_t^p) \leftarrow \{o_i \mid o_i \in \mathcal{O}_t^+,\ \mathcal{R}_i \cap \mathcal{O}_t^+ \neq \emptyset\}. \tag{5}$$

In other words, the set of allowed operations consists of all operations that are not scheduled yet, and at least one related operation is not yet scheduled either. Scheduling an operation o_i means that o_i has to be processed before all operations $o_j \in \mathcal{R}_i \cap \mathcal{O}^+$. If $\mathcal{N}(s_t^p)$ is empty, the remaining operations in $\mathcal{O}_t^+$ are mutually unrelated and a feasible solution is unambiguously defined. Therefore, the remaining operations are appended to s_t^p in any order. Using the mechanism of list scheduler algorithms ensures the feasibility of the constructed solutions.

The transition probabilities for choosing an operation $o_i \in \mathcal{N}(s_t^p)$ at each construction step t are a function of the pheromone values (see Definition 1) and the weights assigned by a weighting function, which are in ACO algorithms called the *heuristic information*. The weighting function that we used is defined by

$$\eta(o_i) \leftarrow \frac{\frac{1}{t_{\mathrm{es}}(o_i, s_t^p)+1}}{\sum_{o_k \in \mathcal{N}(s_t^p)} \frac{1}{t_{\mathrm{es}}(o_k, s_t^p)+1}}, \quad \forall o_i \in \mathcal{N}(s_t^p). \tag{6}$$

Then, the transition probabilities are defined by

$$\mathbf{p}(o_i \mid \mathcal{T}, \eta) \leftarrow \frac{(\min_{o_j \in R_i \cap \mathcal{O}_t^+} \tau_{ij})^\alpha \eta(o_i)}{\sum_{o_k \in \mathcal{N}(s_t^p)} (\min_{o_j \in R_k \cap \mathcal{O}_t^+} \tau_{kj})^\alpha \eta(o_k)}, \quad \forall o_i \in \mathcal{N}(s_t^p), \tag{7}$$

where α is a parameter for adjusting the importance that is given to the pheromone information, respectively, the heuristic information. The formula above determines the probability for each $o_i \in \mathcal{N}(s_t^p)$ to be proportional to the minimum of the pheromone values between o_i and its related and unscheduled operations. This is a reasonable choice, because if this minimum is low it means that there is at least one related operation left that probably should be scheduled before o_i.

The algorithmic framework of the solution construction is shown in Algorithm 4. Following the solution construction mechanism as outlined above, each ant performs a probabilistic beam search. The only component of Algorithm 3 that does not appear in the algorithmic framework of beam search as shown in Algorithm 2 is the procedure $\mathsf{ReduceToRelated}(\mathcal{N}(s_t^p), o_i)$. This procedure is used to restrict set $\mathcal{N}(s_t^p)$ after the first extension of a partial solution s_t^p was performed. Assuming that the first extension of s_t^p was done by adding operation o_i, the restriction is done as follows:

$$\mathcal{N}(s_t^p) \leftarrow \{o_j \in \mathcal{N}(s_t^p) \mid o_j \in \mathcal{R}_i\}. \tag{8}$$

[8] Remember that $\mathcal{R}_i$ is the set of operations that are related to operation o_i.

[9] Consider for example the partial solution $\langle o_1, o_2, o_3, o_4, o_5, o_6 \rangle$ to the problem instance that is shown in Fig. 2. With respect to this partial solution, operation o_7 is not scheduled yet and all the operations that are related to o_7 are already scheduled. Therefore, the place of operation o_7 in the final permutation does not change the final solution, because it will in any way be the last operation on its machine and the last operation of its job.

This means that all further extensions of s_t^p have to be performed with operations that are related to the operation that was chosen for the first extension. The reason is that we want to ensure that all the different extensions of a partial solution result in different feasible solutions when completed. In other words, this restriction avoids building the same solution more than once.[10] Accordingly, in the first construction step there are $|\mathcal{O}|$ possible extensions and in every further construction step there are maximally $(\max_{\mathcal{M}_i \in \mathcal{M}} |\mathcal{M}_i| + \max_{\mathcal{J}_j \in \mathcal{J}} |\mathcal{J}_j|)$ allowed extensions of the current partial solution.

Algorithm 4. Beam-ACO solution construction for the OSS problem

```
input: an empty partial solution s_1^p = ⟨⟩, beam width k_bw, max. number of extensions k_ext
B ← {s_1^p}, B_c ← ∅, t ← 1
while B ≠ ∅ do
  B_ext ← ∅
  for s_t^p ∈ B do
    count ← 1
    N(s_t^p) ← PreSelect(N(s_t^p))
    while count ⩽ k_ext AND N(s_t^p) ≠ ∅ do
      Choose o_i ∈ N(s_t^p) with transition probability p(o_i | T, η) {see Equation 7}
      s_{t+1}^p ← extend s_t^p by appending operation o_i
      N(s_t^p) ← N(s_t^p) \ {o_i}
      if N(s_{t+1}^p) ≠ ∅ then
        B_ext ← B_ext ∪ {s_{t+1}^p}
      else
        B_c ← B_c ∪ {s_{t+1}^p}
      end if
      if count = 1 then
        N(s_t^p) ← ReduceToRelated(N(s_t^p), o_i)
      end if
      count ← count + 1
    end while
  end for
  Rank the partial solutions in B_ext using a lower bound LB(·)
  B ← select the min{k_bw, |B_ext|} highest ranked partial solutions from B_ext
  k ← k + 1
end while
output: a set of feasible solutions B_c
```

In order to specify the remaining components of the solution construction process we need to make three design choices as outlined in Section 4: (1) The definition of a lower bound $LB(\cdot)$ to evaluate partial solutions, (2) the setting of the beam width k_{bw} and the maximum number of extensions k_{ext}

[10] Consider for example the empty partial solution $\langle\rangle$ with respect to the example instance shown in Fig. 2. In the first construction step every operation is a candidate for being added to $\langle\rangle$. Now, consider two of the possible extensions, namely $\langle o_1 \rangle$ and $\langle o_4 \rangle$. Note that o_1 and o_4 are not related. Therefore, both partial solutions can be completed (e.g., $\langle o_1, o_4, o_2, \ldots, o_{10} \rangle$ and $\langle o_4, o_1, o_2, \ldots, o_{10} \rangle$) such that the final permutations are the same solutions. This is avoided by using function $\mathsf{ReduceToRelated}(\mathcal{N}(s_t^p), o_i)$.

of a partial solution, and (3) the specification of a pre-selection mechanism for filtering the set of solution components that can be used to extend a partial solution. In the following we focus on these three design decisions.

The lower bound $LB(\cdot)$: In the following we denote the operation of a job $\mathcal{J}_j \in \mathcal{J}$ that was taken latest into a partial schedule s_t^p by $o^{\mathcal{J}_j}$. Similarly, we denote the operation of a machine $\mathcal{M}_i \in \mathcal{M}$ that was taken latest into the partial schedule s_t^p by $o^{\mathcal{M}_i}$. Furthermore, the partition of the set of operations $\mathcal{O}$ with respect to a partial solution s_t^p into $\mathcal{O}_t^-$ (the operations that are already scheduled) and $\mathcal{O}_t^+$ (the operations that still have to be dealt with) induces a partition of the operations of every job $\mathcal{J}_j \in \mathcal{J}$ into $\mathcal{J}_{j_t}^-$ and $\mathcal{J}_{j_t}^+$ and of every machine $\mathcal{M}_i \in \mathcal{M}$ into $\mathcal{M}_{i_t}^-$ and $\mathcal{M}_{i_t}^+$. Then, the lower bound $LB(\cdot)$ is for a partial solution s_t^p computed as follows:

$$LB(s_t^p) \leftarrow \max\{X, Y\}, \tag{9}$$

where

$$X = \max_{\mathcal{J}_j \in \mathcal{J}} \left\{ t_{\mathrm{ec}}(o^{\mathcal{J}_j}, s_t^p) + \sum_{o \in \mathcal{J}_{j_t}^+} p(o) \right\}, \tag{10}$$

$$Y = \max_{\mathcal{M}_i \in \mathcal{M}} \left\{ t_{\mathrm{ec}}(o^{\mathcal{M}_i}, s_t^p) + \sum_{o \in \mathcal{M}_{i_t}^+} p(o) \right\}, \tag{11}$$

Therefore, lower bound $LB(\cdot)$ consists of summing for every job and machine the processing times of the unscheduled operations, adding the earliest completion time of the operation of the respective job or machine that was scheduled last, and taking the maximum of all these numbers. As all the necessary numbers can be obtained and updated during the construction process, this lower bound can be very efficiently computed.

The pre-selection mechanism PreSelect($\mathcal{N}(s_t^p)$): For filtering set $\mathcal{N}(s_t^p)$ we can use the mechanisms that are available to restrict set $\mathcal{O}_t^+$ at each step in the list scheduler algorithm. There are basically two ways of restricting set $\mathcal{O}_t^+$. The one proposed by Giffler and Thompson [29] works as shown in Algorithm 5. First, the minimal earliest completion time t^* of all the operations in $\mathcal{O}_t^+$ is calculated. Then, one of the machines $\mathcal{M}^*$ with minimal earliest completion time is chosen and $\mathcal{O}_t^+$ is restricted to all operations that need to be processed on machine $\mathcal{M}^*$ and whose earliest possible starting time is smaller than t^*. This way of restricting set $\mathcal{O}_t^+$ produces active schedules.[11]

Algorithm 5. Giffler and Thompson mechanism for restricting $\mathcal{O}_t^+$
input: s_t^p, $\mathcal{O}_t^+$
Determine $t^* \leftarrow \min\{t_{\mathrm{ec}}(o, s_t^p) \mid o \in \mathcal{O}_t^+\}$
$\mathcal{M}^* \leftarrow$ Select randomly from $\{\mathcal{M}_i \in \mathcal{M} \mid \mathcal{M}_{i_t}^+ \cap \neq \emptyset,\ \exists o \in \mathcal{M}_{i_t}^+ \text{ with } t_{\mathrm{ec}}(o, s_t^p) = t^*\}$
$\mathcal{O}_t^+ \leftarrow \{o \in \mathcal{O}_t^+ \mid o \in \mathcal{M}^* \text{ and } t_{\mathrm{es}}(o, s_t^p) < t^*\}$
output: restricted set $\mathcal{O}_t^+$

[11] The set of active schedules is a subset of the set of feasible schedules. An optimal solution is guaranteed to be an active schedule.

Algorithm 6. Non-delay mechanism for restricting $\mathcal{O}_t^+$
input: s_t^p, $\mathcal{O}_t^+$
Determine $t^* \leftarrow \min\{t_{es}(o, s_t^p) \mid o \in \mathcal{O}_t^+\}$
$\mathcal{O}_t^+ \leftarrow \{o \in \mathcal{O}_t^+ \mid t_{es}(o, s_t^p) = t^*\}$
output: restricted set $\mathcal{O}_t^+$

The second major way of restricting set $\mathcal{O}_t^+$ is the non-delay mechanism that is shown in Algorithm 6. First, the earliest possible starting time t^* among all operations in $\mathcal{O}_t^+$ is determined. Then $\mathcal{O}_t^+$ is restricted to all operations that can start at time t^*. By this way of restricting set $\mathcal{O}_t$, non-delay schedules [12] are generated.

After restricting set $\mathcal{O}_t^+$, the restriction of $\mathcal{N}(s_t^p)$ is achieved by removing all operations o_i with $o_i \notin \mathcal{O}_t^+$. Based on these two ways of restricting set $\mathcal{O}_t^+$ we decided to explore the following 4 pre-selection mechanisms: (1) No restriction of $\mathcal{O}_t^+$ at all (henceforth denoted by NR), (2) restriction of $\mathcal{O}_t^+$ due to Giffler and Thompson (henceforth denoted by GT), (3) restriction of $\mathcal{O}_t^+$ by the non-delay method (henceforth denoted by ND), and (4) a combination of (2) and (3) that is achieved by choosing at each construction step randomly between (2) and (3) for restricting $\mathcal{O}_t^+$ (henceforth denoted by GT-ND).

Strategies for setting k_{bw} and k_{ext}: In order to find out if rather high or rather low settings of k_{bw} are required, we decided to test two different settings of k_{bw}, which both depend on the problem instance size. These settings are $k_{bw} = |\mathcal{O}|$, and $k_{bw} = \max\{1, \lfloor |\mathcal{O}|/10 \rfloor\}$. Furthermore, we decided to test two different strategies for setting k_{ext}, the maximal number of extensions of a partial solution. In the first strategy, k_{ext} is set to half of the number of possible extensions of a partial solution s_t^p. Therefore, the setting is $k_{ext} = \max\{1, \lfloor |\mathcal{N}(s_t^p)|/2 \rfloor\}$. Note that this setting depends at each construction step on the current partial solution. The second strategy is based on an idea from limited discrepancy search (LDS) [33]. The idea is that a constructive mechanism that is guided by some policy is more likely to make wrong decisions at early stages of the construction process rather than later. Based on this idea we set k_{ext} to $|\mathcal{N}(s_t^p)|$ for construction steps $t \leqslant \max\{1, \lfloor |\mathcal{O}|/20 \rfloor\}$. Once t has passed this limit, we set k_{ext} to 2. This limit is quite arbitrary. However, it is our aim to find out if this idea works, rather than to find the optimal limit. The first strategy for setting k_{ext} is in the following denoted by MED, whereas the second strategy is denoted by LDS.

It is interesting to note that if the beam width k_{bw} and the maximal number of extensions k_{ext} are big enough, Beam-ACO-OSS is an (inefficient) enumeration method. On the other extreme, if k_{bw} and k_{ext} are set to 1, Beam-ACO-OSS is a standard ACO algorithm where each ant constructs one solution per iteration.

6. Experimental evaluation

All the results that we present in this section were obtained on PCs with AMD Athlon 1100 Mhz CPU running under Linux. The software was developed in C++ (gcc version 2.96). Furthermore, Beam-ACO-OSS is based on the same implementation (i.e., the data structures and the local search) as Standard-ACO-OSS, which was proposed in [15].

[12] The set of non-delay schedules is a subset of the set of active schedules.

6.1. Benchmark instances

There are three different sets of OSS benchmark instances available in the literature. The first set consists of 60 problem instances provided by Taillard [34] (denoted by tai_*). The smallest of these instances consist of 16 operations (4 jobs and 4 machines), and the biggest instances consist of 400 operations (20 jobs and 20 machines). Furthermore, we performed tests on 35 of the difficult OSS instances provided by Brucker et al. [35] (denoted by j*). The number after the letter j denotes the size of the instance. So for example the j5-* instances are on 5 jobs and 5 machines, and the j8-* instances on 8 jobs and 8 machines. Finally, we applied Beam-ACO-OSS (as well as Standard-ACO-OSS, which was not applied to this benchmark set before) to the 80 benchmark instances provided by Guéret and Prins [36] (denoted by gp*). The size of these instances ranges from 3 jobs and 3 machines to 10 jobs and 10 machines. Also this third benchmark set was generated in order to be difficult to solve.

6.2. Parameter settings

In general, many parameters in Beam-ACO-OSS may be considered for parameter tuning. However, we decided to focus on the parameters of the solution construction, rather than on the parameters of the ACO framework. Therefore, we adopted the parameter settings $\rho = 0.1$, and $\alpha = 10$ (see Eq. 7) from Standard-ACO-OSS. Furthermore, we applied Beam-ACO-OSS in all experiments with only one ant per iteration. This is reasonable, as one ant—in contrast to ants in standard ACO algorithms –constructs a number of solutions that depends on the beam width k_{bw}.

Recall that there are three parameters to be set in the solution construction mechanism of Beam-ACO-OSS. We have to decide the beam width k_{bw} ($|\mathcal{O}|$ or $|\mathcal{O}|/10$). Then, we have to decide between the two strategies MED and LDS for setting the maximal extension number k_{ext}. A third parameter is the pre-selection mechanism, where we have the four options GT, GT-ND, ND and NR as outlined in the previous section. Therefore, we have two parameters with two possible values each, and one parameter with four possible values. We tested every combination of the parameter values (16 different settings) on two different OSS benchmark instances: j7-per0-0, a difficult instance from [35], and tai_10x10_1 from [34]. The results are shown in numerical form in Table 1, and in graphical form in Fig. 3.

The results allow us to draw the following conclusions: In general, strategy LDS for setting the maximal extension number k_{ext} outperforms strategy MED on both problem instances. Second, the use of beam width $k_{\text{bw}} = |\mathcal{O}|$ in general outperforms the setting $k_{\text{bw}} = \lfloor |\mathcal{O}|/10 \rfloor$. For the pre-selection mechanisms that we used we can observe that GT and GT-ND work well for problem instance j7-per0-0, whereas GT-ND and ND work very well and much better than the other two settings for problem instance tai_10x10_1. This confirms an observation from [15], that for solving the problem instances by Taillard [34] a strong use of the non-delay mechanism is crucial. Based on these results we chose the parameter settings $k_{\text{bw}} = |\mathcal{O}|$, LDS, and GT-ND for all further experiments.

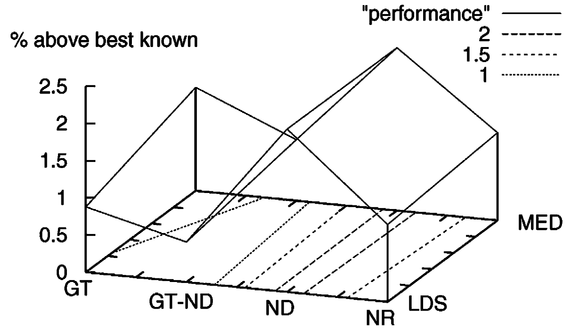
Furthermore, we wanted to explore the influence that the pheromone update and the local search have on the performance of Beam-ACO-OSS. Therefore, we tested four versions of Beam-ACO-OSS on several problem instances. In the following, the notation Beam-ACO-OSS without any further specifications refers to the algorithm version with pheromone update and local search. The four versions that we tested are (1) Beam-ACO-OSS, (2) Beam-ACO-OSS without pheromone update,

Table 1
The table gives the results of Beam-ACO-OSS with different parameter settings (as specified in the first column) for two different OSS benchmark instances

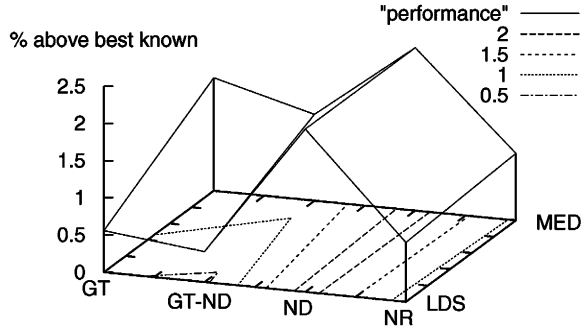
Parameter specification	Beam-ACO-OSS		Rank
	Average	$\bar{t}$	
(a) *Results for instance* j7-per0-0. *Time limit*: 490 s			
GT, LDS, $k_{bw} = \|\mathcal{O}\|$	1053.9	130.409	3
GT, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1057.25	315.194	6
GT, MED, $k_{bw} = \|\mathcal{O}\|$	1063.95	247.205	12
GT, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1062.6	181.918	11
GT-ND, LDS, $k_{bw} = \|\mathcal{O}\|$	1052.3	275.541	1
GT-ND, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1053.7	198.789	2
GT-ND, MED, $k_{bw} = \|\mathcal{O}\|$	1060.15	228.788	9
GT-ND, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1056.55	184.168	5
ND, LDS, $k_{bw} = \|\mathcal{O}\|$	1071	14.881	14.5
ND, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1071	30.879	14.5
ND, MED, $k_{bw} = \|\mathcal{O}\|$	1071	55.11	14.5
ND, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1071	61.243	14.5
NR, LDS, $k_{bw} = \|\mathcal{O}\|$	1056.35	277.12	4
NR, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1058.9	233.741	8
NR, MED, $k_{bw} = \|\mathcal{O}\|$	1057.45	215.865	7
NR, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	1060.35	248.196	10
Results for instance tai_10x10_1. *Time limit*: 100 s			
GT, LDS, $k_{bw} = \|\mathcal{O}\|$	646.049	55.36	10
GT, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	655.649	52.652	14
GT, MED, $k_{bw} = \|\mathcal{O}\|$	654.2	52.301	13
GT, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	658.799	46.231	16
GT-ND, LDS, $k_{bw} = \|\mathcal{O}\|$	637.35	37.842	3
GT-ND, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	639.5	48.223	5
GT-ND, MED, $k_{bw} = \|\mathcal{O}\|$	641.549	40.359	7
GT-ND, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	643.75	50.969	8
ND, LDS, $k_{bw} = \|\mathcal{O}\|$	637	10.856	1
ND, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	637.2	21.554	2
ND, MED, $k_{bw} = \|\mathcal{O}\|$	639.299	40.71	4
ND, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	640.1	47.964	6
NR, LDS, $k_{bw} = \|\mathcal{O}\|$	645.799	28.41	9
NR, LDS, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	649.399	53.384	12
NR, MED, $k_{bw} = \|\mathcal{O}\|$	648.149	49.725	11
NR, MED, $k_{bw} = \lfloor \|\mathcal{O}\|/10 \rfloor$	656.049	50.853	15

The second column of each table provides the average of the best solution values obtained in 20 runs of the algorithm. The third column gives the average computation time that was needed to obtain the best solution of a run. Finally, in the last column of each table, we have ranked the different parameter settings according to the average of the best solution values that are given in the second column.

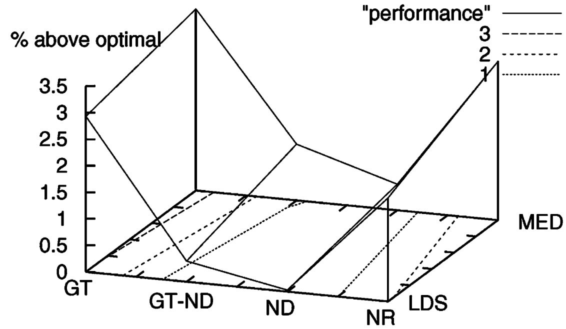
(3) Beam-ACO-OSS without local search, and (4) Beam-ACO-OSS without pheromone update and without local search. The results are shown in the two subtables of Table 2. First, we applied the four versions of Beam-ACO-OSS to the nine instances j7-* of the set of benchmark instances by



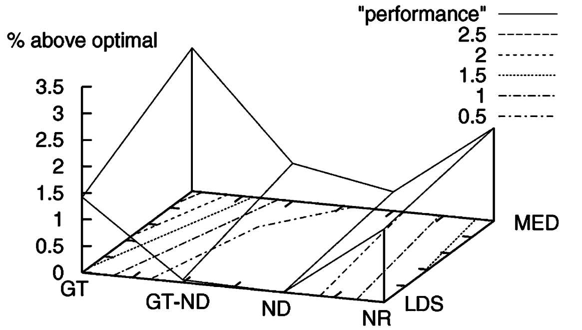
(a) Instance j7-per0-0, $k_{bw} = \lfloor |\mathcal{O}|/10 \rfloor$



(b) Instance j7-per0-0, $k_{bw} = |\mathcal{O}|$



(c) Instance tai_10x10_1, $k_{bw} = \lfloor |\mathcal{O}|/10 \rfloor$



(d) Instance tai_10x10_1, $k_{bw} = |\mathcal{O}|$

Fig. 3. The x-axis and the y-axis define different parameter settings of Beam-ACO-OSS. On the x-axis are the pre-selection mechanisms, on the y-axis the strategies for setting k_{ext}, and the z-axis shows the percentage above the best known solution value of the average solution quality obtained in 20 runs of Beam-ACO-OSS with the parameter settings as specified on the x- and y-axis. (a) and (b) show the results for instance j7-per0-0 (490 s per run), which is a difficult OSS problem instance from the benchmark set provided by Brucker et al. [35]. The results in (a) are obtained with $k_{bw} = \lfloor |\mathcal{O}|/10 \rfloor$, whereas the results in (b) are obtained with $k_{bw} = |\mathcal{O}|$. (c) and (d) show in the same way the results for instance tai_10x10_1 (100 s per run) from the benchmark set provided by Taillard [34]. The legends of the four graphs indicate the "height" of the contour lines given on the x-y plane (i.e., give a contour map of the performance surface).

Brucker et al. [35]. The results are shown in Table 2(a). The first observation is that although the two versions with pheromone update are always better or equal to the two versions without pheromone update, the differences between all four versions are quite small. This indicates the power of using a probabilistic beam search mechanism to construct solutions in the way that we have proposed it. In a second series of experiments we applied the four versions of Beam-ACO-OSS to the ten problem instances gp_* of the benchmark set by Guéret and Prins [36] (see Table 2(b)). Again, the results show that the algorithm versions that use pheromone update are—with the exceptions of gp10-05 in terms of average solution quality obtained and of gp10-09 in terms of the best solution value found—slightly better than the other two versions. Furthermore, it is interesting to note that Beam-ACO-OSS without using the local search procedures seems to have slight advantages over Beam-ACO-OSS. However, as this is not the case for the more difficult j7-* instances we decided for the local search procedures to remain in Beam-ACO-OSS.

Table 2
Results that show the influence of the pheromone update and the local search on the performance of Beam-ACO-OSS

	Beam-ACO-OSS			No update			No LS			No update, no LS		
Instance	Best	Average	$\bar{t}$	Best	Average	$\bar{t}$	Best	Average	$\bar{t}$	Best	Average	$\bar{t}$
(a) *Results for instances* j7-*. *Time limit*: 490 s.												
j7-per0-0	1048	1052.7	207.861	1050	1055.5	195.352	**1048**	1052.65	195.159	1050	1054.1	198.427
j7-per0-1	**1057**	1057.8	91.525	1058	1060.65	180.417	**1057**	1057.8	204.527	1057	1062.5	250.786
j7-per0-2	**1058**	1058.95	175.811	1058	1060.6	283.548	1058	1059.25	200.015	1058	1061.5	221.754
j7-per10-0	**1013**	1016.7	217.538	1016	1020.6	193.229	**1013**	1016.7	213.639	1017	1021.65	196.848
j7-per10-1	**1000**	1002.45	189.808	1005	1008	194.152	1001	1003.9	163.473	1006	1008.65	257.29
j7-per10-2	**1016**	1019.4	180.616	1019	1020.9	193.668	1016	1019.75	180.274	1016	1021.6	278.139
j7-per20-0	**1000**	1000	0.376	**1000**	1000	0.464	**1000**	1000	0.343	**1000**	1000	0.44
j7-per20-1	**1005**	1007.6	259.014	1006	1012.05	235.128	1006	1009.35	231.489	1009	1013.85	297.337
j7-per20-2	**1003**	1007.25	257.234	1004	1009.4	146.411	1004	1006.65	308.064	1006	1009.85	179.933
(b) *Results for instances* gp10_*. *Time limit*: 1000 s.												
gp10-01	**1099**	1108.95	567.483	1108	1117.95	405.615	1101	1110.65	670.663	1108	1117.35	372.649
gp10-02	1101	1107.4	501.614	1101	1110.85	452.031	**1099**	1106.4	448.307	1102	1113.1	497.755
gp10-03	1082	1097.95	658.609	1090	1107.65	504.711	**1081**	1098.35	657.958	1090	1106.15	419.832
gp10-04	1093	1096.6	588.077	1094	1098.75	547.384	**1089**	1096.85	482.128	1094	1098.5	428.987
gp10-05	1083	1092.4	636.339	1082	1091.5	494.88	**1080**	1092.55	644.123	1084	1096.5	510.382
gp10-06	1088	1104.55	595.407	1079	1104.25	455.487	**1079**	1100.65	504.417	1096	1104.7	622.282
gp10-07	**1084**	1091.45	389.504	1087	1096.65	617.419	1087	1092.25	586.355	1087	1098.1	473.85
gp10-08	1099	1104.8	615.811	1098	1105.25	477.025	**1098**	1104.45	675.911	1100	1108.2	527.71
gp10-09	1121	1128.7	554.427	**1120**	1129.4	441.591	1121	1130.85	469.464	1127	1131.5	619.756
gp10-10	**1097**	1106.65	562.495	1102	1111.35	516.68	1098	1107.6	502.204	1100	1109.6	517.052

The table is organized as follows: the first column specifies the problem instance. Then, there are three columns for each of the four different versions of Beam-ACO-OSS, that are (1) Beam-ACO-OSS with pheromone update and local search, (2) without pheromone update, (3) without local search, and (4) without update and without local search. The first of the three respective columns gives the best solution found in 20 runs. An objective function value is in bold, if it beats the other three algorithm versions. In case of ties the average solution quality decides. The second column gives the average of the best solutions found in 20 runs, and the third column gives the average CPU time that was needed to find the best solutions in the 20 runs.

6.3. Results

The state-of-the-art metaheuristics for the OSS problem are on one side the hybrid genetic algorithm by Liaw [16] (henceforth denoted by GA-Liaw), and the best of the genetic algorithms from the paper [17] by Prins (henceforth denoted by GA-Prins). However, GA-Liaw was only applied to the Taillard instances and to a subset of the Brucker instances, whereas GA-Prins was applied to all available OSS benchmark instances. These are the algorithms—besides Standard-ACO-OSS [15]—to which we compare Beam-ACO-OSS.

We compare Beam-ACO-OSS to GA-Liaw and GA-Prins only in terms of the best solution values found. The reason is that both GA-Liaw and GA-Prins were only applied once to each problem instance. Therefore, average results do not exist for these two approaches. In addition, for the following reasons the computation times of Beam-ACO-OSS are not directly comparable to the computation times of GA-Liaw and GA-Prins. Firstly, the meaning of the computation times is different. While Beam-ACO-OSS is stopped with respect to a maximum computation time limit, GA-Liaw is stopped when either a maximum number of iterations is reached or the optimal solution is found, whereas GA-Prins is stopped when either a maximum number if iterations is reached, 12 000 iterations are performed without improvement, or if a lower bound is reached. Accordingly, the reported computation times have a different meaning. While for Beam-ACO-OSS we report for each problem instance the average time when the best solution value of a run was found, Liaw and Prins report on the time their algorithm was stopped. Secondly, computation times for GA-Prins were given in [17] only as averages over sets of benchmark instances which include very small as well as very big instances.

Furthermore, the comparison of the computation times might not be very meaningful due to potentially quite different implementations of the algorithms and different computational platforms. While the results of Beam-ACO-OSS were obtained from a C++ programme running under Linux on a 1100 MHz PC, the results of GA-Prins as reported in [17] were obtained from a Turbo Pascal programme running under Windows on a 166 MHz PC, and the results of GA-Liaw as reported in [16] were obtained from a C programme running on a 266 MHz PC. However, in general the computation times of all three algorithms are quite low. To our opinion the performance of an algorithm is therefore of higher importance. (i.e., as long as the computation times are within a few minutes it is to our opinion reasonable to prefer an algorithm that obtains better solution qualities).

Our test results are shown in Tables 3–5. The format of these tables is as follows: In the first column we give the name of the problem instance. In the second column we give the best objective function value that is known for the corresponding instance. Brackets refer to the fact that the value is *not* proved to be the optimal solution value. Furthermore, if there is a right-to-left arrow pointing to the best known value, it means that this value was improved by either Beam-ACO-OSS or Standard-ACO-OSS. Then, there are two columns where we give the best solution values found by GA-Liaw, respectively, GA-Prins. Furthermore, there are twice four columns for displaying the results of Beam-ACO-OSS, respectively, Standard-ACO-OSS. In the first one of these four columns we give the value of the best solution found in 20 runs of the algorithm. The second one gives the average of the values of the best solutions found in 20 runs. In the third column we give the standard deviation of the average given in the second column, and in column 4 we note the average time that was needed to find the best solutions in the 20 runs. Finally, the last column of each of the three tables gives the CPU time limit for Beam-ACO-OSS and Standard-ACO-OSS. A value in a column about the best solution values found is indicated in bold, if it is the best in the comparison between Beam-ACO-OSS and Standard-ACO-OSS. Ties are broken by using the average solution quality as a second criterion. Finally, an asterisk marking a value refers to the fact that this value is equal to the best known solution value for the respective problem instance.

We have set the CPU time limits for Beam-ACO-OSS when applied to the Taillard instances to $|\mathcal{O}|$ seconds, and to $10 \cdot |\mathcal{O}|$ seconds when applied to the Brucker et al. instances, respectively the Guéret and Prins instances. The reason for allowing a higher time limit for the application to the latter instances is that they were designed to be more difficult to solve than the Taillard instances. The only exception to this scheme are the CPU time limits for the application to the Taillard instances on 25, respectively, 49, operations, where we have set the CPU time limit to $2 \cdot |\mathcal{O}|$ seconds, because historically most of the existing approaches had more difficulties to solve these instances in comparison to the other Taillard instances.

Table 3
Results for the OSS benchmark instances provided by Taillard [34].

Instance	Best known	GA-Liaw [14]	GA-Prins [17]	Beam-ACO-OSS				Standard-ACO-OSS [15]				Time limits(s)
				Best	Average	$\sqrt{\sigma}$	$\bar{t}$	Best	Average	$\sqrt{\sigma}$	$\bar{t}$	
tai_4x4_1	193	*193	*193	***193**	193	0	0.095	*193	193.199	0.615	5.08	16
tai_4x4_2	236	*236	239	***236**	236	0	0.105	*236	238.25	1.332	1.381	16
tai_4x4_3	271	*271	*271	***271**	271	0	0.177	***271**	271	0	0.603	16
tai_4x4_4	250	*250	*250	***250**	250	0	0.291	*250	250.4	0.82	5.984	16
tai_4x4_5	295	*295	*295	***295**	295	0	0.043	***295**	295	0	2.771	16
tai_4x4_6	189	*189	*189	***189**	189	0	0.061	***189**	189	0	0.385	16
tai_4x4_7	201	*201	*201	***201**	201	0	0.152	*201	201.099	0.447	5.958	16
tai_4x4_8	217	*217	*217	***217**	217	0	0.008	***217**	217	0	0.02	16
tai_4x4_9	261	*261	*261	***261**	261	0	0.025	***261**	261	0	0.127	16
tai_4x4_10	217	*217	221	***217**	217	0	0.814	***217**	217	0	2.216	16
tai_5x5_1	300	*300	301	***300**	300	0	2.129	*300	300.399	0.502	19.623	50
tai_5x5_2	262	*262	263	***262**	262	0	0.398	*262	262.899	1.372	22.513	50
tai_5x5_3	323	*323	335	***323**	323	0	0.904	*323	328.55	2.91	16.409	50
tai_5x5_4	310	*310	316	***310**	310	0	13.014	311	312.85	1.225	21.824	50
tai_5x5_5	326	*326	330	***326**	326	0	2.037	*326	329.1	1.41	15.469	50
tai_5x5_6	312	*312	*312	***312**	312	0	2.167	***312**	312	0	1.816	50
tai_5x5_7	303	*303	308	***303**	303	0	1.146	*303	305.35	1.755	9.816	50
tai_5x5_8	300	*300	304	***300**	300	0	3.128	301	301.85	0.988	16.127	50
tai_5x5_9	353	*353	358	***353**	353	0	3.854	356	356	0	10.075	50
tai_5x5_10	326	*326	328	***326**	326	0	0.795	*326	327.3	0.978	10.502	50
tai_7x7_1	435	*435	436	***435**	435	0	2.044	*435	436.55	1.276	51.353	98
tai_7x7_2	443	*443	447	***443**	443	0	19.133	*443	446.55	1.276	36.683	98
tai_7x7_3	468	*468	472	***468**	468	0	15.944	471	474.85	2.433	36.59	98
tai_7x7_4	463	*463	*463	***463**	463	0	1.601	*463	464.449	1.316	51.732	98
tai_7x7_5	416	*416	417	***416**	416	0	2.291	*416	416.05	0.223	33.182	98
tai_7x7_6	451	*451	455	***451**	451.35	0.745	24.794	455	455.85	1.871	46.768	98
tai_7x7_7	422	*422	426	***422**	422.149	0.489	22.942	424	427.1	2.149	46.64	98
tai_7x7_8	424	*424	*424	***424**	424	0	1.128	*424	424.3	0.47	47.821	98
tai_7x7_9	458	*458	*458	***458**	458	0	1.058	***458**	458	0	30.932	98
tai_7x7_10	397	*398	*398	***398**	398	0	1.592	*398	398.149	0.489	25.861	98
tai_10x10_1	637	*637	*637	***637**	637.35	0.587	40.709	*637	642.549	3.017	53.113	100
tai_10x10_2	588	*588	*588	***588**	588	0	2.934	*588	589.149	1.182	57.908	100
tai_10x10_3	598	*598	*598	***598**	598	0	27.81	599	603.85	2.3	61.51	100
tai_10x10_4	577	*577	*577	***577**	577	0	2.571	*577	577.25	0.55	32.374	100
tai_10x10_5	640	*640	*640	***640**	640	0	8.515	*640	643.149	1.98	54.22	100
tai_10x10_6	538	*538	*538	***538**	538	0	2.504	*538	538.1	0.307	23.507	100
tai_10x10_7	616	*616	*616	***616**	616	0	5.188	*616	617.85	1.694	33.579	100
tai_10x10_8	595	*595	*595	***595**	595	0	14.923	*595	598.049	1.959	32.198	100
tai_10x10_9	595	*595	*595	***595**	595	0	5.043	*595	596	1.376	58.235	100
tai_10x10_10	596	*596	*596	***596**	596	0	7.448	*596	598.299	1.976	52.249	100
tai_15x15_1	937	*937	*937	***937**	937	0	14.205	*937	937.549	0.825	90.608	225
tai_15x15_2	918	*918	*918	***918**	918	0	21.033	*918	919.95	1.316	120.623	225

Table 3 (*continued*)

Instance	Best known	GA-Liaw [14]	GA-Prins [17]	Beam-ACO-OSS				Standard-ACO-OSS [15]				Time limits(s)
				Best	Average	$\sqrt{\sigma}$	$\bar{t}$	Best	Average	$\sqrt{\sigma}$	$\bar{t}$	
tai_15x15_3	871	*871	*871	***871**	871	0	14.258	*871	871.45	0.998	86.208	225
tai_15x15_4	934	*934	*934	***934**	934	0	14.17	*934	934.149	0.366	76.606	225
tai_15x15_5	946	*946	*946	***946**	946	0	24.699	*946	947.299	1.26	101.29	225
tai_15x15_6	933	*933	*933	***933**	933	0	16.511	*933	933.299	0.571	84.331	225
tai_15x15_7	891	*891	*891	***891**	891	0	20.945	*891	893.299	2.105	141.311	225
tai_15x15_8	893	*893	*893	***893**	893	0	14.192	*893	893.1	0.307	61.067	225
tai_15x15_9	899	*899	*899	***899**	899.649	1.136	104.09	902	907.1	2.77	137.908	225
tai_15x15_10	902	*902	*902	***902**	902	0	18.083	*902	904.45	2.235	129.507	225
tai_20x20_1	1155	*1155	*1155	***1155**	1155	0	54.805	*1155	1157.3	1.38	215.132	400
tai_20x20_2	1242	1241	*1241	***1241**	1241	0	79.646	1247	1248.8	1.794	191.11	400
tai_20x20_3	1257	*1257	*1257	***1257**	1257	0	48.527	*1257	1257.4	0.68	179.83	400
tai_20x20_4	1248	*1248	*1248	***1248**	1248	0	49.025	*1248	1248.25	0.55	229.548	400
tai_20x20_5	1256	*1256	*1256	***1256**	1256	0	49.073	*1256	1256.65	0.988	197.81	400
tai_20x20_6	1204	*1204	*1204	***1204**	1204	0	49.292	*1204	1205.7	1.08	174.52	400
tai_20x20_7	1294	*1294	*1294	***1294**	1294	0	64.963	1296	1299.75	2.048	214.879	400
tai_20x20_8	1169 ←	1177	1171	***1169**	1170.25	1.482	227.825	1177	1180.95	2.163	189.277	400
tai_20x20_9	1289	*1289	*1289	***1289**	1289	0	48.594	*1289	1289.35	0.587	193.639	400
tai_20x20_10	1241	*1241	*1241	***1241**	1241	0	48.787	*1241	1241.15	0.366	122.736	400

For an explanation of the table format see Section 6.3.

6.3.1. *Results for the Taillard instances (Table 3)*

The results show a clear advantage of Beam-ACO-OSS over Standard-ACO-OSS. Beam-ACO-OSS is the first algorithm that solves all the Taillard instances to optimality.[13] The only problem instance that was to our knowledge not solved before is tai_20x20_8, and Beam-ACO-OSS is the first algorithm that solves this problem instance. Beam-ACO-OSS obtains for 55 of the 60 problem instances a standard deviation of 0, and in comparison to Standard-ACO-OSS we observe a substantial reduction in CPU time. On the small problem instances (tai_4x4_*) Beam-ACO-OSS is about 10 times faster than Standard-ACO-OSS, whereas on the biggest problem instances (tai_20x20_*) Beam-ACO-OSS is about twice as fast as Standard-ACO-OSS.

Concerning the comparison of Beam-ACO-OSS to the two GA algorithms, we can observe that GA-Liaw is only slightly worse than Beam-ACO-OSS on this benchmark set. GA-Liaw solves 58 of the 60 benchmark instances to optimality. However, it was not able to solve two of the largest benchmark instances. Furthermore, GA-Prins is clearly inferior to the other methods on this benchmark set, with major difficulties to solve the problem instances tai_5x5_* and tai_7x7_*.

[13] We can be sure of this due to the fact that all the results obtained by Beam-ACO-OSS are equal to the values of lower bounds.

Table 4
Results for the OSS benchmark instances provided by Brucker et al. [35]

Instance	Best known	GA-Liaw [16]	GA-Prins [17]	Beam-ACO-OSS Best	Average	$\sqrt{\sigma}$	$\bar{t}$	Standard-ACO-OSS [15] Best	Average	$\sqrt{\sigma}$	$\bar{t}$	Time limit(s)
j5-per0-0	1042	*1042	1050	***1042**	1042	0	0.298	***1042**	1042	0	35.526	250
j5-per0-1	1054	*1054	*1054	***1054**	1054	0	0.036	***1054**	1054	0	0.181	250
j5-per0-2	1063	*1063	1085	***1063**	1063	0	0.633	***1063**	1063	0	39.913	250
j5-per10-0	1004	*1004	*1004	***1004**	1004	0	0.799	***1004**	1004	0	10.492	250
j5-per10-1	1002	*1002	*1002	***1002**	1002	0	9.162	*1002	1003.65	1.531	65.026	250
j5-per10-2	1006	*1006	*1006	***1006**	1006	0	0.195	***1006**	1006	0	25.519	250
j5-per20-0	1000	*1000	1004	***1000**	1000	0	0.1	***1000**	1000	0	7.435	250
j5-per20-1	1000	*1000	*1000	***1000**	1000	0	0.042	***1000**	1000	0	0.116	250
j5-per20-2	1012	*1012	*1012	***1012**	1012	0	0.639	***1012**	1012	0	0.192	250
j6-per0-0	1056	*1056	1080	***1056**	1056	0	27.331	1061	1076.25	6.348	100.176	360
j6-per0-1	1045	*1045	*1045	*1045	1049.7	5.027	61.246	***1045**	1048.7	4.366	157.417	360
j6-per0-2	1063	*1063	1079	***1063**	1063	0	38.786	1070	1077.4	3.377	59.74	360
j6-per10-0	1005	*1005	1016	***1005**	1005	0	10.563	*1005	1020.3	5.202	120.515	360
j6-per10-1	1021	*1021	1036	***1021**	1021	0	11.253	*1021	1022.8	4.708	141.272	360
j6-per10-2	1012	*1012	*1012	***1012**	1012	0	1.346	***1012**	1012	0	3.891	360
j6-per20-0	1000	*1000	1018	***1000**	1003.6	1.231	31.027	1004	1008.15	3.013	122.246	360
j6-per20-1	1000	*1000	*1000	***1000**	1000	0	0.703	***1000**	1000	0	32.749	360
j6-per20-2	1000	*1000	1001	***1000**	1000	0	3.817	***1000**	1000	0	17.108	360
j7-per0-0	(1048)	1063	1071	***1048**	1052.7	2.386	207.861	1070	1071.5	2.039	231.353	490
j7-per0-1	1055	1058	1076	**1057**	1057.8	0.41	91.525	1069	1071.2	1.239	130.989	490
j7-per0-2	1056	1059	1082	**1058**	1058.95	1.276	175.811	1070	1075.25	2.244	196.487	490
j7-per10-0	1013	1022	1036	***1013**	1016.7	2.451	217.538	1034	1036.05	1.571	163.698	490
j7-per10-1	1000	1014	1010	***1000**	1002.45	2.459	189.808	1006	1006	0	33.213	490
j7-per10-2	1011	1020	1035	**1016**	1019.4	2.01	180.616	1032	1032.95	1.503	196.708	490
j7-per20-0	1000	*1000	*1000	***1000**	1000	0	0.376	***1000**	1000	0	3.371	490
j7-per20-1	1005	1011	1030	***1005**	1007.6	2.233	259.014	1015	1016.55	1.394	156.09	490
j7-per20-2	1003	1010	1020	***1003**	1007.25	2.124	257.234	1011	1014.25	3.058	180.62	490
j8-per0-1	(1039) ←	N.a.	1075	***1039**	1048.65	6.515	313.404	1065	1074.35	4.591	311.913	640
j8-per0-2	(1052) ←	N.a.	1073	***1052**	1057.05	2.981	323.343	1065	1076.25	6.163	301.906	640
j8-per10-0	(1020) ←	N.a.	1053	***1020**	1026.9	5.046	346.408	1036	1043.3	4.52	355.615	640
j8-per10-1	(1004) ←	N.a.	1029	***1004**	1012.4	3.604	308.802	1022	1026.35	2.978	308.052	640
j8-per10-2	(1009) ←	N.a.	1027	***1009**	1013.65	4.307	399.35	1020	1028.4	5.04	350.731	640
j8-per20-0	1000	N.a.	1015	***1000**	1001	1.213	237.162	1003	1010.45	3.086	345.139	640
j8-per20-1	1000	N.a.	*1000	***1000**	1000	0	2.526	***1000**	1000	0	37.536	640
j8-per20-2	1000 ←	N.a.	1014	***1000**	1000.55	1.145	286.136	1001	1006.9	2.971	283.031	640

For an explanation of the table format see Section 6.3.

6.3.2. Results for the Brucker et al. instances (Table 4)

As a result of the relatively low difficulty of the Taillard instances, the Brucker et al. instances were generated in order to be more difficult to solve. The increased difficulty results in an increased performance difference between Beam-ACO-OSS and Standard-ACO-OSS, respectively, Beam-ACO-OSS

Table 5
Results for the OSS benchmark instances provided by Guéret and Prins in [36]

Instance	Best known	GA-Liaw	GA-Prins	Beam-ACO-OSS				Standard-ACO-OSS [15]				Time limit(s)
				Best	Average	$\sqrt{\sigma}$	$\bar{t}$	Best	Average	$\sqrt{\sigma}$	$\bar{t}$	
gp03-01	1168	N.a.	*1168	***1168**	1168	0	0	***1168**	1168	0	0	90
gp03-02	1170	N.a.	*1170	***1170**	1170	0	0	***1170**	1170	0	0	90
gp03-03	1168	N.a.	*1168	***1168**	1168	0	0	***1168**	1168	0	0	90
gp03-04	1166	N.a.	*1166	***1166**	1166	0	0	***1166**	1166	0	0	90
gp03-05	1170	N.a.	*1170	***1170**	1170	0	0	***1170**	1170	0	0	90
gp03-06	1169	N.a.	*1169	***1169**	1169	0	0	***1169**	1169	0	0	90
gp03-07	1165	N.a.	*1165	***1165**	1165	0	0	***1165**	1165	0	0	90
gp03-08	1167	N.a.	*1167	***1167**	1167	0	0	***1167**	1167	0	0	90
gp03-09	1162	N.a.	*1162	***1162**	1162	0	0	***1162**	1162	0	0	90
gp03-10	1165	N.a.	*1165	***1165**	1165	0	0	***1165**	1165	0	0	90
gp04-01	1281	N.a.	*1281	***1281**	1281	0	0.015	***1281**	1281	0	0.043	160
gp04-02	1270	N.a.	*1270	***1270**	1270	0	0.031	***1270**	1270	0	0.096	160
gp04-03	1288	N.a.	*1288	***1288**	1288	0	0.014	***1288**	1288	0	0.004	160
gp04-04	1261	N.a.	*1261	***1261**	1261	0	0.025	***1261**	1261	0	0.325	160
gp04-05	1289	N.a.	*1289	***1289**	1289	0	0.009	***1289**	1289	0	0.009	160
gp04-06	1269	N.a.	*1269	***1269**	1269	0	0.01	***1269**	1269	0	0.008	160
gp04-07	1267	N.a.	*1267	***1267**	1267	0	0.107	***1267**	1267	0	0.612	160
gp04-08	1259	N.a.	*1259	***1259**	1259	0	0.017	***1259**	1259	0	0.007	160
gp04-09	1280	N.a.	*1280	***1280**	1280	0	0.079	*1280	1283	1.777	15.369	160
gp04-10	1263	N.a.	*1263	***1263**	1263	0	0.018	***1263**	1263	0	0.011	160
gp05-01	1245	N.a.	*1245	***1245**	1245	0	1.065	***1245**	1245	0	0.303	250
gp05-02	1247	N.a.	*1247	***1247**	1247	0	1.125	***1247**	1247	0	0.081	250
gp05-03	1265	N.a.	*1265	***1265**	1265	0	0.298	***1265**	1265	0	0.212	250
gp05-04	1258	N.a.	*1258	*1258	1258.6	1.095	10.32	***1258**	1258.1	0.307	88.377	250
gp05-05	1280	N.a.	*1280	***1280**	1280	0	0.28	***1280**	1280	0	0.335	250
gp05-06	1269	N.a.	*1269	*1269	1269.05	0.223	9.279	***1269**	1269	0	0.635	250
gp05-07	1269	N.a.	*1269	***1269**	1269	0	0.083	***1269**	1269	0	0.104	250
gp05-08	1287	N.a.	*1287	***1287**	1287	0	0.12	***1287**	1287	0	0.159	250
gp05-09	1262	N.a.	*1262	***1262**	1262	0	1.401	***1262**	1262	0	0.401	250
gp05-10	1254	N.a.	*1254	*1254	1254.6	0.502	6.068	***1254**	1254.2	0.41	50.247	250
gp06-01	1264	N.a.	*1264	***1264**	1264.65	0.489	30.733	*1264	1264.75	0.444	53.03	360
gp06-02	1285	N.a.	*1285	*1285	1285.65	0.489	48.643	***1285**	1285	0	13.58	360
gp06-03	(1255)	N.a.	*1255	***1255**	1255	0	29.678	*1255	1255.2	0.41	120.159	360
gp06-04	1275	N.a.	*1275	***1275**	1275	0	25.896	***1275**	1275	0	2.989	360
gp06-05	1299	N.a.	1300	*1299	1299.15	0.366	39.848	***1299**	1299	0	8.086	360
gp06-06	1284	N.a.	*1284	***1284**	1284	0	42.912	***1284**	1284	0	112.65	360
gp06-07	(1290)	N.a.	*1290	***1290**	1290	0	10.439	***1290**	1290	0	5.687	360
gp06-08	1265	N.a.	1266	*1265	1265.2	0.41	71.846	***1265**	1265	0	87.269	360
gp06-09	(1243)	N.a.	*1243	***1243**	1243	0	9.797	*1243	1243.1	0.307	61.804	360
gp06-10	(1254)	N.a.	*1254	***1254**	1254	0	4.257	***1254**	1254	0	23.295	360
gp07-01	(1159)	N.a.	*1159	***1159**	1159	0	86.9	***1159**	1159	0	19.931	490
gp07-02	(1185)	N.a.	*1185	***1185**	1185	0	80.233	***1185**	1185	0	1.284	490

Table 5 (*continued*)

Instance	Best known	GA-Liaw	GA-Prins	Beam-ACO-OSS				Standard-ACO-OSS [15]				Time limit(s)
				Best	Average	$\sqrt{\sigma}$	$\bar{t}$	Best	Average	$\sqrt{\sigma}$	$\bar{t}$	
gp07-03	1237	N.a.	*1237	***1237**	1237	0	40.869	***1237**	1237	0	14.446	490
gp07-04	(1167)	N.a.	*1167	***1167**	1167	0	59.104	***1167**	1167	0	7.823	490
gp07-05	1157	N.a.	*1157	***1157**	1157	0	124.4	***1157**	1157	0	60.921	490
gp07-06	(1193)	N.a.	*1193	*1193	1193.9	0.447	152.316	***1193**	1193	0	41.692	490
gp07-07	1185	N.a.	*1185	*1185	1185.05	0.223	91.087	***1185**	1185	0	3.307	490
gp07-08	(1180) ←	N.a.	1181	*1180	1181.35	1.136	206.618	***1180**	1180.1	0.307	171.775	490
gp07-09	(1220)	N.a.	*1220	*1220	1220.05	0.223	127.896	***1220**	1220	0	79.751	490
gp07-10	1270	N.a.	*1270	*1270	1270.05	0.223	65.521	***1270**	1270	0	0.824	490
gp08-01	1130 ←	N.a.	1160	***1130**	1132.4	0.82	334.999	1131	1132.2	1.105	253.686	640
gp08-02	(1135) ←	N.a.	1136	***1135**	1136.1	0.64	228.379	1136	1138.55	3.268	173.644	640
gp08-03	1110 ←	N.a.	1111	1111	1113.65	1.531	336.249	***1110**	1115.4	3.56	223.165	640
gp08-04	(1154) ←	N.a.	1168	***1154**	1156	2.152	275.667	*1154	1167.3	3.13	117.341	640
gp08-05	1218	N.a.	*1218	1219	1219.75	0.786	347.652	***1218**	1218	0	84.219	640
gp08-06	(1116) ←	N.a.	1128	***1116**	1123.15	6.368	359.165	1117	1129.9	4.678	261.855	640
gp08-07	(1126)	N.a.	1128	***1126**	1134.6	4.333	296.764	*1126	1135.9	5.739	247.259	640
gp08-08	(1148)	N.a.	*1148	*1148	1148.95	1.986	277.328	***1148**	1148.5	2.236	193.339	640
gp08-09	1114	N.a.	1120	1117	1118.95	2.163	278.971	***1114**	1114.85	0.587	216.003	640
gp08-10	(1161)	N.a.	*1161	*1161	1161.5	0.76	281.21	***1161**	1161	0	115.317	640
gp09-01	(1135)←	N.a.	1143	***1135**	1142.75	3.905	412.859	1146	1147.9	0.447	92.369	810
gp09-02	(1112) ←	N.a.	1114	***1112**	1113.65	1.268	430.72	*1112	1115.75	2.971	339.173	810
gp09-03	(1117) ←	N.a.	1118	1118	1120.35	3.183	427.901	***1117**	1117.8	0.41	229.681	810
gp09-04	1130 ←	N.a.	1131	***1130**	1139.95	5.031	549.605	1138	1140.55	0.998	423.673	810
gp09-05	1180	N.a.	*1180	*1180	1180.5	0.76	295.81	***1180**	1180	0	33.129	810
gp09-06	(1093) ←	N.a.	1117	***1093**	1095.55	1.791	386.963	1096	1115.5	4.696	368.055	810
gp09-07	(1097) ←	N.a.	1119	***1097**	1101.35	4.221	431.358	1115	1116.95	1.538	466.172	810
gp09-08	(1106) ←	N.a.	1110	***1106**	1113.7	4.168	376.168	1108	1110.15	1.089	399.477	810
gp09-09	(1126) ←	N.a.	1132	1127	1132.45	5.185	402.6	***1126**	1127.55	2.928	492.055	810
gp09-10	(1120) ←	N.a.	1130	***1120**	1126.3	5.016	435.747	1122	1127.8	3.138	360.788	810
gp10-01	(1099) ←	N.a.	1113	***1099**	1108.95	6.893	567.483	1108	1114.25	4.81	488.887	1000
gp10-02	(1099) ←	N.a.	1120	**1101**	1107.4	5.968	501.614	1102	1112	6.44	518.934	1000
gp10-03	(1081) ←	N.a.	1101	**1082**	1097.95	9.167	658.609	1097	1104.8	3.188	522.292	1000
gp10-04	(1089) ←	N.a.	1090	1093	1096.6	2.798	588.077	***1089**	1094.3	3.246	499.312	1000
gp10-05	(1080) ←	N.a.	1094	**1083**	1092.4	6.459	636.339	1091	1096.65	4.246	399.796	1000
gp10-06	(1072) ←	N.a.	1074	1088	1104.55	6.336	595.407	***1072**	1078.4	10.772	443.577	1000
gp10-07	(1081) ←	N.a.	1083	1084	1091.45	5.623	389.504	***1081**	1082.45	1.145	483.911	1000
gp10-08	(1098)	N.a.	*1098	1099	1104.8	3.721	615.811	**1099**	1104.3	3.435	575.089	1000
gp10-09	(1120)←	N.a.	1121	**1121**	1128.7	3.743	554.427	1124	1128.15	3.116	617.443	1000
gp10-10	(1092) ←	N.a.	1095	1097	1106.65	7.895	562.495	***1092**	1094.4	1.465	412.624	1000

For an explanation of the table format see Section 6.3. Note that the improved best known solutions for instances gp10-02, gp10-03, gp10-05, and gp10-09 were obtained by different versions of Beam-ACO-OSS (see Table 2(b)).

and the two GA algorithms. This becomes especially apparent on the bigger problem instances j7-* and j8-*. Beam-ACO-OSS is clearly the best algorithm. It finds for 9 of the 17 biggest instances the best known solution values, and is able to improve the best known solution values for further 5 of the remaining 8 biggest problem instances. In contrast, Standard-ACO-OSS and GA-Prins only find the best known solution values for 2 of the 17 biggest problem instances. Furthermore, GA-Liaw is consistently better than GA-Prins on the instances to which it was applied. However, it was not applied to the biggest problem instances (j8-*), and only finds for 1 of the 9 instances j7-* the best known solution value.

6.3.3. Results for the Guéret and Prins instances (Table 5)

Also the Guéret and Prins instances were generated in order to be difficult to solve. Beam-ACO-OSS and Standard-ACO-OSS (which was applied for the first time to this set of benchmark instances) improve for 24 of the 80 instances the best known solution values: Beam-ACO-OSS improves the best known solution value of 14 instances, Standard-ACO-OSS improves the best known solution value of 8 instances, and both algorithms find the same improved solution value for further 2 instances. The advantage of Beam-ACO-OSS over Standard-ACO-OSS is not as clear on this benchmark set as on the other two benchmark sets. However, both algorithms clearly outperform GA-Prins, which was the best algorithm so far for this benchmark set. It is interesting to note that for one of the instances (i.e., gp10-06) Standard-ACO-OSS is clearly better than Beam-ACO-OSS. This possibly indicates that the lower bound that is used in Beam-ACO-OSS leads the algorithm for this problem instance to a "wrong" area in the search space. This is also indicated by the average computation time on some of the instances. For example, Standard-ACO-OSS needs on average 0.824 s for solving problem instance gp07-10 in 20 out of 20 runs, whereas Beam-ACO-OSS needs on average 65.521 seconds for solving this instance in 19 out of 20 runs.

To summarize, we can state that Beam-ACO-OSS is a new state-of-the-art algorithm for solving the existing OSS benchmark instances. Altogether it was able to improve the best known solution values for 22 of the available benchmark instances (1 Taillard instance, 5 Brucker instances, and 16 Guéret and Prins instances). Furthermore, we were able to substantially improve on the results obtained by the best standard ACO algorithm for the OSS problem (Standard-ACO-OSS).

7. Conclusions and outlook

In this paper we have hybridized the solution construction mechanism of ACO algorithms with BS. The resulting way of constructing solutions can be regarded as a probabilistic BS procedure. This approach, which we called Beam-ACO, is general and can in principle be applied to any CO problem. Furthermore, we proposed a Beam-ACO approach for the application to open shop scheduling, which is an NP-hard scheduling problem. We experimentally showed that the results obtained by the Beam-ACO approach improve on the results that are obtained by the currently best standard ACO algorithm that is available for OSS. Furthermore, we showed that the Beam-ACO approach is even a state-of-the-art method for solving the existing OSS benchmark instances. This was done by comparing the Beam-ACO approach to the two best approaches that are currently available in the literature.

Encouraged by these results we plan to apply Beam-ACO approaches to other CO problems. To our opinion, Beam-ACO approaches are especially promising for the application to problems where tree search methods perform well and where it is difficult to find a well-working neighborhood for local search-based methods.

Acknowledgements

Christian Blum acknowledges support by the “Metaheuristics Network”, a Research Training Network funded by the Improving Human Potential program of the CEC, Grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community’s opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] Papadimitriou CH, Steiglitz K. Combinatorial optimization—algorithms and complexity. New York: Dover Publications; 1982.
[2] Ginsberg M. Essentials of artificial intelligence. San Mateo, CA: Morgan Kaufmann Publishers; 1993.
[3] Aarts E, Lenstra JK, editors. Local search in combinatorial optimization. Series in Discrete Mathematics and Optimization. Chichester, UK: Wiley; 1997.
[4] Pinedo M. Scheduling: theory, algorithms, and systems. Englewood Cliffs: Prentice-Hall; 1995.
[5] Glover F, Kochenberger G, editors. Handbook of metaheuristics. Boston, MA: Kluwer Academic Publishers; 2003.
[6] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 2003;35(3):268–308.
[7] Glover F, Laguna M. Tabu search. Boston, MA: Kluwer Academic Publishers; 1997.
[8] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220(4598):671–80.
[9] Lourenço HR, Martin O, Stützle T. Iterated local search, In: Glover F, Kochenberger G, editors, Handbook of metaheuristics, International series in operations research & management Science, vol. 57, Norwell, MA: Kluwer Academic Publishers; 2002. p. 321–53.
[10] Resende MGC, Ribeiro CC. Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics. Boston, MA: Kluwer Academic Publishers; 2003. p. 219–50.
[11] Focacci F, Laburthe F, Lodi A. Local search and constraint programming. In: Glover F, Kochenberger G, editors, Handbook of metaheuristics, Boston, MA, Kluwer Academic Publishers; 2003, p. 369–404.
[12] Dorigo M, Di Caro G. The ant colony optimization meta-heuristic. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. London, UK: McGraw Hill; 1999. p. 11–32.
[13] Dorigo M, Stützle T. Ant colony optimization. Boston, MA: MIT Press; 2004.
[14] Ow PS, Morton TE. Filtered beam search in scheduling. International Journal of Production Research 1988;26: 297–307.
[15] Blum C. An ant colony optimization algorithm to tackle shop scheduling problems Technical Report TR/IRIDIA/2003-01, IRIDIA, Université Libre de Bruxelles, Belgium; 2003.
[16] Liaw C-F. A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research 2000;124:28–42.
[17] Prins C. Competitive genetic algorithms for the open-shop scheduling problem. Mathematical Methods of Operations Research 2000;52(3):389–411.
[18] Maniezzo V. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS Journal on Computing 1999;11(4):358–69.

[19] Maniezzo V, Milandri M. An ant-based framework for very strongly constrained problems. In: Dorigo M, Di Caro G, Sampels M, editors. Proceedings of ANTS2002—From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms. Lecture Notes in Computer Science, vol. 2463. Berlin, Germany: Springer; 2002. p. 222–7.
[20] Birattari M, Di Caro G, Dorigo M. Toward a formal foundation of ant programming. In: Dorigo M, Di Caro G, Sampels M, editors. Proceedings of ANTS 2002—Third International Workshop on Ant Algorithms. Lecture Notes in Computer Science, vol. 2463. Berlin, Germany: Springer; 2002. p. 188–201.
[21] Bertsekas DP. Dynanmic programming and optimal control, vol. 1. Belmont, MA: Athena Scientific; 1995.
[22] Dorigo M, Maniezzo V, Colorni A. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy; 1991.
[23] Dorigo M. Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy; 1992. 140p. (in Italian)
[24] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics—Part B 1996;26(1):29–41.
[25] den Besten ML, Stützle T, Dorigo M. Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In: Proceedings of EvoStim'01, Lecture Notes in Computer Science, Berlin: Springer; 2001, p. 441–52.
[26] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation 2000;6(4):333–46.
[27] Colorni A, Dorigo M, Maniezzo V, Trubian M. Ant system for job-shop scheduling. Belgian Journal of Operations Research, Statistics and Computer Science 1993;34(1):39–54.
[28] Sampels M, Blum C, Mastrolilli M, Rossi-Doria O. Metaheuristics for group shop scheduling. In: Merelo Guervós JJ, et al., editor, Proceedings of PPSN-VII, Seventh International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science, vol. 2439. Berlin, Germany: Springer; 2002. p. 631–40.
[29] Giffler B, Thompson GL. Algorithms for solving production scheduling problems. Operations Research 1960;18: 487–503.
[30] Blum C, Roli A, Dorigo M. HC-ACO: The hyper-cube framework for ant colony optimization. In: Proceedings of MIC'2001—Meta-heuristics International Conference, vol. 2, Porto, Portugal; 2001, p. 399–403.
[31] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man and Cybernetics—Part B; to appear. Also available as Technical Report TR/IRIDIA/2003-03, IRIDIA, Université Libre de Bruxelles, Belgium; 2003.
[32] Stützle T, Hoos HH. $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. Future Generation Computer Systems 2000;16(8):889–914.
[33] Harvey WD, Ginsberg ML. Limited discrepancy search. In: Mellish CS, editor, Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95, vol. 1. Montréal, Québec, Canada, Los Altos, CA: Morgan Kaufmann; 1995, p. 607–15.
[34] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operations Research 1993;64:278–85.
[35] Brucker P, Hurink J, Jurisch B, Wöstmann B. A branch & bound algorithm for the open-shop problem. Discrete Applied Mathematics 1997;76:43–59.
[36] Guéret C, Prins C. A new lower bound for the open-shop problem. Annals of Operations Research 1999;92:165–83.