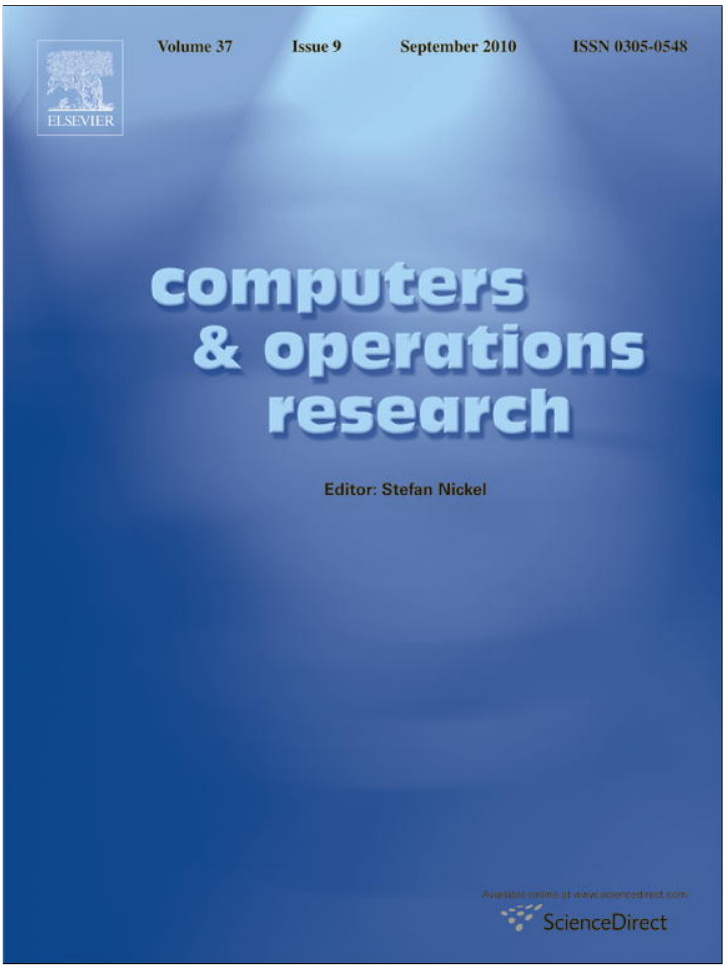

Computers & Operations Research 37 (2010) 1570–1583

Contents lists available at ScienceDirect

Computers & Operations Research

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

Beam-ACO for the travelling salesman problem with time windows ☆

Manuel López-Ibáñez [a,*], Christian Blum [b]

[a] IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
[b] ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain

ARTICLE INFO

Available online 3 December 2009

Keywords:
Ant colony optimization
Travelling salesman problem with time windows
Hybridization

ABSTRACT

The travelling salesman problem with time windows is a difficult optimization problem that arises, for example, in logistics. This paper deals with the minimization of the travel-cost. For solving this problem, this paper proposes a Beam-ACO algorithm, which is a hybrid method combining ant colony optimization with beam search. In general, Beam-ACO algorithms heavily rely on accurate and computationally inexpensive bounding information for differentiating between partial solutions. This work uses stochastic sampling as a useful alternative. An extensive experimental evaluation on seven benchmark sets from the literature shows that the proposed Beam-ACO algorithm is currently a state-of-the-art technique for the travelling salesman problem with time windows when travel-cost optimization is concerned.

1. Introduction

The travelling salesman problem with time windows (TSPTW) is an important problem in logistics. More specifically, it can be used for modelling routing as well as scheduling tasks. Concerning routing, it models the problem of finding an efficient route to visit a number of customers, starting and ending at a depot, with the added difficulty that each customer must be visited within a given time window. The TSPTW can also model the problem of scheduling jobs on a single machine where the setup time of each job depends on the previous job, and each job has a release time and a deadline. In the context of the routing problem the travel cost is the objective most often minimized, whereas in the context of the scheduling problem the makespan is usually subject to optimization. In this work we focus on the routing problem under travel-cost optimization. We will henceforth refer to this problem simply as the TSPTW. The TSPTW is proven to be $\mathcal{NP}-$hard, and even finding a feasible solution is an $\mathcal{NP}-$complete problem [1]. Moreover, the TSPTW is closely related to a number of important problems. For example, while the travelling salesman problem (TSP) is a special case of the TSPTW, the TSPTW itself can be seen as a special case of the vehicle routing problem with time windows (VRPTW) when only one vehicle is concerned.

1.1. History

Early works [2,3] focused on makespan optimization. The proposed techniques are based on branch-and-bound and solve instances with up to 50 nodes. However, they are not able to handle time windows that are wide or mostly overlapping. Most later works deal with travel-cost optimization. Langevin et al. [4] considered both makespan and travel-cost optimization. They describe a two-commodity flow formulation within a branch-and-bound scheme being able to solve instances with up to 40 nodes. Dumas et al. [5] extended earlier dynamic programming approaches by using state space reduction techniques that enable the solution of instances with up to 200 customers. More recently, Ascheuer et al. [6] considered a branch-and-cut algorithm applying techniques tailored for the asymmetric TSPTW. Balas and Simonetti [7] proposed a linear-time dynamic programming algorithm for various TSP variants with precedence constraints including the TSPTW. Constraint programming has also been applied to develop exact methods [8,9].

Because of the inherent difficulty of the TSPTW, heuristic techniques have been considered as well. Carlton and Barnes [10] developed a tabu search approach that allows the examination of infeasible neighbors through the implementation of a (static) penalty function. Gendreau et al. [11] presented a construction and post-optimization heuristic. Calvo [12] presented a construction heuristic that starts with a solution to an ad hoc assignment problem, proceeds with a greedy insertion procedure to obtain a complete solution and applies local search to further improve the solution. Recently, Ohlmann and Thomas [13] proposed a compressed annealing (CA) algorithm, a variant of simulated annealing [14] that makes use of a variable penalty method. In

☆This work was supported by Grant TIN2007-66523 (FORMALISM) of the Spanish government. Moreover, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

* Corresponding author.
E-mail addresses: manuel.lopez-ibanez@ulb.ac.be (M. López-Ibáñez). cblum@lsi.upc.edu (C. Blum).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.11.015

their excellent paper they provide an extensive comparison with previous approaches. Their approach can currently be regarded as state of the art.

1.2. Our contribution

In this work, we propose a Beam-ACO algorithm [15,16] for solving the TSPTW. This algorithm results from combining the metaheuristic ant colony optimization [17] with the tree search method beam search [18]. Due to the lack of an efficient and effective lower bound, which is needed by the beam search component, we employ instead stochastic sampling [19,20] for the evaluation of partial solutions. This paper is a significant extension of previous work [21,22]. First, we add a sophisticated local search method for improving the solutions constructed by Beam-ACO. Second, we apply our algorithm to all benchmark sets that can be found in the literature. More specifically, we use the five benchmark sets considered by Ohlmann and Thomas [13], and additionally we consider two more benchmark sets. For each benchmark set we compare to the best available algorithms. To the best of our knowledge, this is the most comprehensive comparison of algorithms for the TSPTW to date. Apart from the extensive comparison to existing approaches, we also present a study of the influence of different algorithmic components on the performance of the algorithm. In particular, we examine the influence of the pheromone information, the effect of different degrees of stochastic sampling, and how the algorithm behavior changes when local search is incorporated.

1.3. Organization

This work is organized as follows. In Section 2 we give a technical description of the TSPTW. Section 3 introduces the Beam-ACO algorithm to tackle the TSPTW. In Section 4 we describe the experimental evaluation, and in Section 5 we offer conclusions and an outlook to future work.

2. The TSP with time windows

Given an undirected complete graph $G=(N,A)$—where $N=\{0,1,\ldots,n\}$ is a set of nodes representing the depot (node 0) and n customers, and $A=N\times N$ is the set of edges connecting the nodes—a solution to the TSPTW is a tour visiting each node once, starting and ending at the depot. Hence, a tour is represented as $P=(p_0=0,p_1,\ldots,p_n,p_{n+1}=0)$, where the subsequence $(p_1,\ldots,p_k,\ldots,p_n)$ is a permutation of the nodes in $N\backslash\{0\}$ and p_k denotes the index of the customer at the kth position of the tour. Two additional elements, $p_0=0$ and $p_{n+1}=0$, represent the depot where each tour must start and end.

For every edge $a_{ij}\in A$ between two nodes i and j, there is an associated cost $c(a_{ij})$. This cost typically represents the travel time between customers i and j, plus a service time at customer i. Furthermore, there is a time window $[e_i,l_i]$ associated to each node $i\in N$, which specifies that customer i cannot be serviced before e_i or visited later than l_i. In most formulations of the problem, waiting times are permitted, that is, a node i can be reached before the start of its time window e_i, but cannot be left before e_i. Therefore, given a particular tour P, the departure time from customer p_k is calculated as $D_{p_k}=\max(A_{p_k},e_{p_k})$, where $A_{p_k}=D_{p_{k-1}}+c(a_{p_{k-1},p_k})$ is the arrival time at customer p_k.

As mentioned before, in this paper we focus on the minimization of the travel cost, that is, the minimization of the cost of the edges traversed along the tour. This objective function has been chosen by the majority of previous works. Given this objective, the TSPTW can be formally defined as follows:

$$\begin{aligned}&\text{minimize}\quad f(P)=\sum_{k=0}^{n}c(a_{p_k,p_{k+1}})\\&\text{subject to}\quad \Omega(P)=\sum_{k=0}^{n+1}\omega(p_k)=0,\end{aligned}\tag{1}$$

where

$$\omega(p_k)=\begin{cases}1 & \text{if } A_{p_k}>l_{p_k},\\ 0 & \text{otherwise,}\end{cases}$$

$$A_{p_{k+1}}=\max(A_{p_k},e_{p_k})+c(a_{p_k,p_{k+1}}).$$

In the above definition, $\Omega(P)$ denotes the number of time window constraints that are violated by tour P, which must be zero for feasible solutions.

3. The Beam-ACO algorithm

In the following we outline the Beam-ACO algorithm that we developed for the TSPTW. As mentioned before, Beam-ACO algorithms are hybrids between ant colony optimization and beam search. Ant colony optimization (ACO) is a metaheuristic that is based on the probabilistic construction of solutions. At each algorithm iteration, a number of solutions are constructed independently of each other. Beam-ACO employs instead at each iteration a probabilistic beam search procedure that constructs a number of solutions interdependently and in parallel. At each construction step, beam search keeps a certain number of the best partial solutions available for further extension. These partial solutions are selected with respect to bounding information. Hence, accurate and inexpensive bounding information is a crucial component of beam search. A problem arises when the bounding information is either misleading or when this information is computationally expensive, which is the case for the TSPTW. In this work we use stochastic sampling [19,20] as an alternative to bounding information. When using stochastic sampling, each partial solution is completed a certain number of times in a stochastic way. The information obtained by these stochastic samples is used to rank the different partial solutions. The worst partial solutions are then excluded from further examination.

First, we focus on the solution construction part of the algorithm, because it is crucial for the success of Beam-ACO. Note that solution construction is necessary for *beam search* as well as for *stochastic sampling*. Both procedures are based on a pheromone model $\mathcal{T}$, which is a finite set of numerical values. In the case of the TSPTW, $\forall a_{ij}\in A$, $\exists\tau_{ij}\in\mathcal{T}$, $0\le\tau_{ij}\le 1$. Being currently at costumer i, τ_{ij} represents the desirability of travelling to unvisited customer j next. In general, the greater the pheromone value τ_{ij}, the greater is the desirability of visiting j next.

One feature that distinguishes our approach from other algorithms from the literature is the fact that we allow the construction of infeasible solutions, and we do not make use of penalty terms. Therefore, it is necessary to define a way of comparing between different—possibly infeasible—solutions. This will be done lexicographically ($<_{lex}$) by first minimizing the number of constraint violations (Ω) and, in the case of an equal number of constraint violations, by comparing the tour cost (f). More formally, we compare two different solutions P and P' as follows:

$$P<_{\text{lex}}P' \iff \Omega(P)<\Omega(P') \text{ or } (\Omega(P)=\Omega(P') \text{ and } f(P)<f(P')).\tag{2}$$

3.1. Stochastic sampling

The process of completing a given partial solution several times in a stochastic way is known in the literature as stochastic sampling. We will make use of this methodology within beam search as explained in Section 3.2.

A partial solution P is completed by adding the unvisited costumers one by one until all costumers are visited. At each step, the set of unvisited costumers is denoted by $\mathcal{N}(P)$. Once all customers have been added to the tour, it is completed by adding node 0 which represents the depot. The decision of which customer to choose at each step is done with the help of pheromone information and heuristic information. This is done by firstly generating a random number q uniformly distributed within $[0,1]$ and comparing this value with a parameter q_0 called the determinism rate. If $q \leq q_0$, $j \in \mathcal{N}(P)$ is chosen deterministically as the costumer with the highest product of pheromone and heuristic information, that is, $j = \text{argmax}_{k \in \mathcal{N}(P)}\{\tau_{ik} \cdot \eta_{ik}\}$, where i is the last customer added to the partial tour P, and η_{ij} is the heuristic information that represents an estimation of the benefit of visiting customer j directly after customer i. Otherwise, j is stochastically chosen from the following distribution of probabilities:

$$\mathbf{p}_i(j) = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{k \in \mathcal{N}(P)} \tau_{ik} \cdot \eta_{ik}} \quad \text{if } j \in \mathcal{N}(P). \tag{3}$$

Regarding the definition of η_{ij}, several existing greedy functions for the TSPTW may be used for that purpose. When deciding which customer should be visited next, not only a small travel cost between customers is desirable but also those customers whose time window finishes sooner should be given priority to avoid constraint violations. In addition, visiting those customers whose time window starts earlier may prevent waiting times. Hence, we use a heuristic information that combines travel cost (c_{ij}), latest service time (l_j) and earliest service time (e_j). Their normalized values are combined as follows:

$$\eta_{ij} = \lambda^c \frac{c^{\max} - c_{ij}}{c^{\max} - c^{\min}} + \lambda^l \frac{l^{\max} - l_j}{l^{\max} - l^{\min}} + \lambda^e \frac{e^{\max} - e_j}{e^{\max} - e^{\min}}, \tag{4}$$

where $\lambda^c + \lambda^l + \lambda^e = 1$ are weights that allow to balance the importance of each type of information. In earlier experiments, we found that no single combination of weights would perform best across all instances of a benchmark set. Therefore, we decided to define the weights randomly for each application of probabilistic beam search. As a consequence, all stochastic sampling actions applied within a specific application of probabilistic beam search use the same weight setting as the corresponding probabilistic beam search.

3.2. Probabilistic beam search

The probabilistic beam search (PBS) that is used within Beam-ACO is described in Algorithm 1. The algorithm requires three input parameters: $k_{\text{bw}} \in \mathbb{Z}^+$ is the *beam width*, $\mu \in \mathbb{R}^+ \geq 1$ is a parameter that is used to derive the number of children that can be chosen at each step, and N^s is the number of *stochastic samples* taken for evaluating a partial solution. Moreover, B_t denotes a set of partial solutions (that is, partial tours) called the *beam*. Hereby, index t denotes the current construction step of beam search. At any time it holds that $|B_t| \leq k_{\text{bw}}$, that is, the number of partial solutions in the beam must be smaller than or equal to the beam width. A problem-dependent greedy function $v()$ is utilized to assign a weight to partial solutions.

Algorithm 1. Probabilistic beam search (PBS) for the TSPTW.

```
B_0 := {(0)}
randomly define the weights λ^c, λ^l, and λ^e
for t := 0 to n do
    C := C(B_t)
    for k := 1, ..., min{⌊μ · k_bw⌋, |C|} do
        ⟨P,j⟩ := ChooseFrom(C)
        C := C \ ⟨P,j⟩
        B_{t+1} := B_t ∪ ⟨P,j⟩
    end for
    B_{t+1} := Reduce(B_{t+1}, k_bw)
end for
output: argmin_lex{T | T ∈ B_n}
```

At the start of the algorithm the beam only contains one partial tour starting at the depot, that is, $B_0 = \{(0)\}$. Let $C = \mathcal{C}(B_t)$ denote the set of all possible extensions of the partial tours in B_t. A partial tour P may be extended by adding a customer j not yet visited by that tour. Such a candidate extension of a partial tour is—in the context of PBS—henceforth denoted by $\langle P,j \rangle$. At each construction step, at most $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ candidate extensions are selected from C by means of the procedure ChooseFrom(C) to form the new beam B_{t+1}.[1] At the end of each step, the new beam B_{t+1} is reduced by means of the procedure Reduce in case it contains more than k_{bw} partial solutions. When $t = n$, that is, when n construction steps have been performed, all partial tours in B_n are completed by adding the depot, and finally the best solution is returned.

The procedure ChooseFrom(C) chooses a candidate extension $\langle P,j \rangle$ from C, either deterministically or probabilistically according to the determinism rate q_0. More precisely, for each call to ChooseFrom(C), a random number q is generated and if $q \leq q_0$ then the decision is taken deterministically by choosing the candidate extension that maximises the product of the pheromone information and the greedy function: $\langle P,j \rangle = \text{argmax}_{\langle P',k \rangle \in C} \tau(\langle P',k \rangle) \cdot v(\langle P',k \rangle)^{-1}$, where $\tau(\langle P',k \rangle)$ corresponds to the pheromone value $\tau_{ik} \in \mathcal{T}$, supposing that i is the last customer visited in tour P'. Otherwise, if $q > q_0$, the decision is taken stochastically according to the following probabilities:

$$\mathbf{p}(\langle P,j \rangle) = \frac{\tau(\langle P,j \rangle) \cdot v(\langle P,j \rangle)^{-1}}{\sum_{\langle P',k \rangle \in C} \tau(\langle P',k \rangle) \cdot v(\langle P',k \rangle)^{-1}}. \tag{5}$$

The greedy function $v(\langle P,j \rangle)$ assigns a heuristic value to each candidate extension $\langle P,j \rangle$. In principle, we might use the greedy function η as given in Eq. (4) for that purpose, that is, $v(\langle P,j \rangle) = \eta(\langle P,j \rangle)$. As in the case of the pheromone information, the notation $\eta(\langle P,j \rangle)$ refers to the value of η_{ij} as defined in Eq. (4), supposing that i is the last customer visited in partial solution P. However, when comparing two extensions $\langle P,j \rangle \in C$ and $\langle P',k \rangle \in C$, the value of η might be misleading in case $P \neq P'$. We solved this problem by defining the greedy function $v()$ as the sum of the ranks of the heuristic information values that correspond to the construction of the extension. For an example see Fig. 1. The edge labels of the search tree are tuples that contain the (fictious) values of the heuristic information (η) in the first place, and the corresponding rank in the second place. For example, the extension 2 of the partial solution (1), denoted by $\langle (1),2 \rangle$ has greedy value $v(\langle (1),2 \rangle) = 1+2 = 3$.

[1] Note that parameter $\mu \geq 1$ was chosen to be a real value in order to allow a fine-grained adjustment of the number of candidate extensions to be chosen at each step of PBS. The choice of $\mu \geq 1$ to be an integer number would only allow multiples of k_{bw}. For example, with $k_{\text{bw}} = 10$, the number of candidate extensions to be chosen were restricted to be in $\{10, 20, 30, \ldots\}$. However, $k_{\text{bw}} = 10$ and a setting of $\mu = 1.5$ allows the selection of 15 candidate extensions, which would not be possible if μ were an integer number.

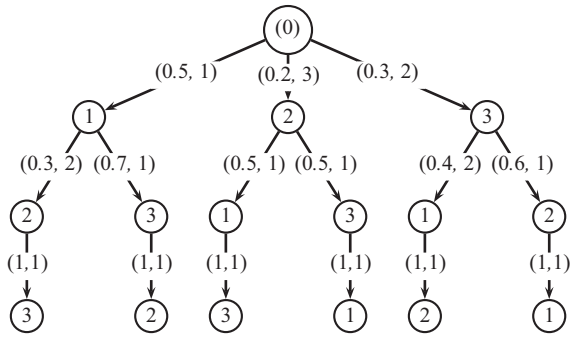

Fig. 1. Search tree corresponding to a problem instance with three customers. Edge labels are tuples that contain the heuristic information (η) in the first place, and the corresponding rank in the second place.

Finally, the application of procedure Reduce removes the worst $\max\{|B_t|-k_{bw}, 0\}$ partial solutions from B_t. As mentioned before, we use *stochastic sampling* for evaluating partial solutions. More specifically, for each partial solution, a number N^s of complete solutions is sampled as explained in Section 3.1. The value of the best of these samples (with respect to Eq. (2) is used for evaluating the corresponding partial solution. Only the k_{bw} best partial solutions (with respect to their corresponding best samples) are kept in B_t and the others are discarded. In our implementation, these best k_{bw} partial solutions are determined by a sorting algorithm that is provided by the programming language that we used (C++). Therefore, we did not have to worry about tie breaking.

3.3. Beam-ACO framework

The probabilistic beam search outlined in the previous section is used within an ACO algorithm implemented in the hyper-cube framework [23]. A high level description of this ACO algorithm is given in Algorithm 2. The main variables used to control the flow of the algorithm are: (1) the *best-so-far* solution P^{bf}, that is, the best solution generated since the start of the algorithm; (2) the *restart-best* solution P^{rb}, that is, the best solution generated since the last restart of the algorithm; (3) the *iteration-best* solution P^{ib}, that is, the best solution constructed in the current iteration, (4) the *convergence factor* (cf), $0 \le cf \le 1$, which is a measure of how far the algorithm is from convergence; and (5) the Boolean variable *bs_update*, which is set to true when the algorithm reaches convergence.

Algorithm 2. Beam-ACO algorithm for the TSPTW.

```
1:  input: N^s, k_bw ∈ Z^+, μ ∈ R^+, q_0 ∈ [0,1] ⊂ R
2:  P^bf := null, P^rb := null, cf := 0, bs_update := false
3:  τ_ij := 0.5  ∀τ_ij ∈ T
4:  while CPU time limit not reached do
5:    P^ib := PBS(k_bw, μ, N^s) //see Algorithm 1
6:    P^ib := LocalSearch(P^ib)
7:    if P^ib <_lex P^rb then P^rb := P^ib
8:    if P^ib <_lex P^bf then P^bf := P^ib
9:    cf := ComputeConvergenceFactor(T)
10:   if bs_update = true and cf > 0.99 then
11:     τ_ij := 0.5  ∀τ_ij ∈ T
12:     P^rb := null, bs_update := false
13:   else
14:     if cf > 0.99 then bs_update := true
15:     ApplyPheromoneUpdate(cf, bs_update, T, P^ib, P^rb, P^bf)
16:   end if
17: end while
18: output: P^bf
```

The algorithm roughly works as follows. First, all variables are initialized. The pheromone values are set to their initial value 0.5. Then, the algorithm iterates a main loop until a maximum CPU time limit is reached. Each iteration consists of the following steps. First, a probabilistic beam search algorithm is executed (as explained in Section 3.2). This produces the iteration-best solution P^{ib}, which is then subject to the application of local search. After updating the best-so-far solution, a new value for the convergence factor cf is computed. Depending on this value, as well as on the value of the Boolean variable *bs_update*, a decision on whether to restart the algorithm or not is made. If the algorithm is restarted, all the pheromone values are reset to their initial value (0.5). The algorithm is iterated until the CPU time limit is reached. Once terminated, the algorithm returns the best solution found which corresponds to P^{bf}. In the following we describe the remaining procedures of Algorithm 2 in more detail.

LocalSearch(P^{ib}): The local search applied in this work is based on the 1-opt neighborhood in which a single customer is removed from the tour and reinserted in a different position. The local search implemented follows the description of Carlton and Barnes [10], although in their description they left out many details. Previous local search approaches made use of a penalty term for evaluating infeasible solutions. Instead we compare solutions lexicographically following Eq. (2).

Algorithm 3. 1-Opt local search for the TSPTW.

```
1:  input: P = (p_0, ..., p_{n+1})
2:  P^best := P
3:  for k := 1 to n−1 do
4:      P' := P
5:      if not is_time_window_infeasible(p'_k, p_{k+1}') then
6:          P' := swap(P', k) // see Algorithm 4
7:          if P' <_lex P^best then P^best := P'
8:          P'' := P'
9:          for d := k+1 to n−1 do
10:             if is_time_window_infeasible(p'_d, p_{d+1}') then break
11:             P' := swap(P', d) // see Algorithm 4
12:             if P' <_lex P^best then P^best := P'
13:         end for
14:         P' := P''
15:         for d := k−1 to 1 do
16:             if is_time_window_infeasible(p'_d, p_{d+1}') then break
17:             P' := swap(P', d) // see Algorithm 4
18:             if P' <_lex P^best then P^best := P'
19:         end for
20:     end if
21: end for
22: output: P^best
```

Algorithm 3 describes the way of choosing the best neighbor of an input solution P within the 1-opt neighborhood. Procedure LocalSearch(P^{ib}) in Algorithm 2 refers to the iterative application of this algorithm until no better solution can be found. Given a starting solution P, all insertion moves of customer p_k into a different position of P are incrementally explored for $k=1,\ldots,n-1$. This is done in two stages. First, all insertions of p_k into later positions of the tour are examined by a sequence of swap moves exchanging customer p_d and p_{d+1}, for $d=k,\ldots,n-1$. Second, all insertions of customer p_{k+1} into an earlier position of the tour are examined. Since, inserting customer p_{k+1} one position earlier is equivalent to inserting customer p_k one position later, the second stage skips the first movement, which was already evaluated in the first stage, and proceeds by a sequence of

swap moves exchanging customers p_d and p_{d+1} for $d=k-1,\ldots,1$. We say that a customer i is *strongly time-window infeasible* with respect to customer j if and only if $e_j+t_{ji}>l_i$, that is, if the earliest time for leaving j plus the travel time from j to i is larger than the latest arrival time at i [10]. If customer i is strongly time-window infeasible with respect to j, then no feasible tour can visit i later than j. In the local search, strong time window infeasibility is taken into account to avoid insertion moves that produce infeasible solutions [10].

In order to provide a reproducible description of the local search that we implemented, we also describe procedure swap() from Algorithm 3 in more detail in Algorithm 4. The tour cost can be evaluated in constant time by calculating the difference in cost of the exchanged arcs. On the other hand, the calculation of the makespan and the number of constraint violations may require adjusting the arrival times after the exchanged arcs. This will not be necessary if there is a positive waiting time at a customer $k>d+2$ both before ($A_k<e_{p_k}$) and after ($A'_k<e_{p_k}$) a swap move is applied to customer p_d. In this case it holds that $A_k=A'_k=e_{p_k}$, and subsequent customers are not affected by the move. It is not clear whether Carlton and Barnes [10] made use of this speed-up in their tabu search approach. Our experiments show that it may notably reduce computation time.

ComputeConvergenceFactor($\mathcal{T}$): This procedure computes the convergence factor cf, which is a function of the current pheromone values, as follows:

$$cf=2\left(\frac{\sum_{\tau_{ij}\in\mathcal{T}}\max\{\tau^{\max}-\tau_{ij},\tau_{ij}-\tau^{\min}\}}{|\mathcal{T}|\cdot(\tau^{\max}-\tau^{\min})}-0.5\right), \tag{6}$$

where $\tau^{\max}$ and $\tau^{\min}$ are, respectively, the maximum and minimum pheromone values allowed. Hence, $cf=0$ when the algorithm is initialized (or reset), that is, when all pheromone values are set to 0.5. In contrast, when the algorithm has converged, then $cf=1$. In all other cases, cf has a value within (0,1).

ApplyPheromoneUpdate(cf, bs_update, $\mathcal{T}$, P^{ib}, P^{rb}, P^{bf}): In general, three solutions are used for updating the pheromone values. These are the *iteration-best* solution P^{ib}, the *restart-best* solution P^{rb}, and the *best-so-far* solution P^{bf}. The influence of each solution on the pheromone update depends on the state of convergence of the algorithm as measured by the convergence factor cf. Hence, each pheromone value $\tau_{ij}\in\mathcal{T}$ is updated as follows:

$$\tau_{ij}=\tau_{ij}+\rho\cdot(\xi_{ij}-\tau_{ij}), \tag{7}$$

Algorithm 4. Procedure swap(P,k) of Algorithm 3.

input: a tour P, a position in the tour k
$f_{old}:=f(P)$
$\Omega_{old}:=\Omega(P)$
$\Delta c:=c(a_{p_{k-1},p_{k+1}})+c(a_{p_{k+1},p_k})+c(a_{p_k,p_{k+2}})-c(a_{p_{k-1},p_k})-c(a_{p_k,p_{k+1}})-c(a_{p_{k+1},p_{k+2}})$
The new objective function value of P is $f_{old}+\Delta c$ (by delta-evaluation)
if $A_{p_k}>l_{p_k}$ **then** $\Omega_{old}:=\Omega_{old}-1$
if $A_{p_{k+1}}>l_{p_{k+1}}$ **then** $\Omega_{old}:=\Omega_{old}-1$
if $A_{p_{k+2}}>l_{p_{k+2}}$ **then** $\Omega_{old}:=\Omega_{old}-1$
$A_{p_k}:=\max(A_{p_{k-1}}+c(a_{p_{k-1},p_{k+1}}),e_{p_{k+1}})$
$A_{p_{k+1}}:=\max(A_{p_k}+c(a_{p_{k+1},p_{k-1}}),e_{p_k})$
$A_{p_{k+2}}:=\max(A_{p_{k+1}}+c(a_{p_k,p_{k+2}}),e_{p_{k+2}})$
if $A_{p_k}>l_{p_{k+1}}$ **then** $\Omega_{old}:=\Omega_{old}+1$
if $A_{p_{k+1}}>l_{p_k}$ **then** $\Omega_{old}:=\Omega_{old}+1$
if $A_{p_{k+2}}>l_{p_{k+2}}$ **then** $\Omega_{old}:=\Omega_{old}+1$
for $i:=k+3$ **to** $n+1$ **do**
    $A'_{p_i}:=A_{p_{i-1}}+c(a_{p_{i-1},p_i})$
    if $A_{p_i}<e_{p_i}$ **then**
        // We had to wait before …
        if $A'_{p_i}<e_{p_i}$ **then**
            $A_{p_i}:=e_{p_i}$ // …we still have to wait …
            break // …so nothing else changes.
        end if
    else
        // We did not wait before …
        if $A_{p_i}>l_{p_i}$ **then** $\Omega_{old}:=\Omega_{old}-1$
        if $A'_{p_i}<e_{p_i}$ **then**
            $A_{p_i}:=e_{p_i}$ // …we wait now …
            next // …so the next customer is affected.
        end if
    end if
    $A_{p_i}:=A'_{p_i}$ // …we do not wait now.
    if $A_{p_i}>l_{p_i}$ **then** $\Omega_{old}:=\Omega_{old}+1$
end for
The new number of constraint violations of P is Ω_{old} (by delta-evaluation)
output: $P:=(p_0,\ldots,p_{k-1},p_{k+1},p_k,p_{k+2},\ldots,p_{n+1})$

Table 1
Setting of κ^{ib}, κ^{rb} and κ^{bf} depending on the convergence factor *cf* and the Boolean control variable *bs_update*.

bs_update	false				true
cf	[0, 0.4)	[0.4,0.6)	[0.6,0.8)	[0.8,1]	–
κ^{ib}	1	2/3	1/3	0	0
κ^{rb}	0	1/3	2/3	1	0
κ^{bf}	0	0	0	0	1

with $\xi_{ij} = \kappa^{ib} \cdot P^{ib}_{ij} + \kappa^{rb} \cdot P^{rb}_{ij} + \kappa^{bf} \cdot P^{bf}_{ij}$, where ρ is a parameter that determines the learning rate, P^{*}_{ij} is 1 if customer j is visited after customer i in solution P^{*} and 0 otherwise, κ^{ib} is the weight (i.e. the influence) of solution P^{ib}, κ^{rb} is the weight of solution P^{rb}, κ^{bf} is the weight of solution P^{bf}, and $\kappa^{ib}+\kappa^{rb}+\kappa^{bf}=1$. For our application we used a standard update schedule as shown in Table 1 and a value of $\rho = 0.1$.

After the pheromone update rule in Eq. (7) is applied, pheromone values that exceed $\tau^{max} = 0.999$ are set back to τ^{max} (similarly for $\tau^{min} = 0.001$). This is done in order to avoid a complete convergence of the algorithm, which is a situation that should be avoided. This completes the description of our Beam-ACO approach for the TSPTW.

4. Experimental evaluation

We implemented Beam-ACO in C++ and conducted all experiments on a computer with an Intel Xeon X3350 processor with 2.66 GHz CPU and 6 MB of cache size running GNU/Linux 2.6.18. In the following we first describe a series of experiments that we conducted for obtaining a better understanding of the influence of different algorithmic components on the performance of Beam-ACO. Afterwards we present an extensive experimental evaluation on seven different sets of benchmark instances from the literature.

4.1. Analysis of algorithmic components

With the aim of obtaining a better understanding of the behavior of Beam-ACO, we conducted a series of experiments with parameters $k_{bw} = 10$, $\mu = 1.5$, $N^{s} = 5$, $q_0 = 0.9$, and a time limit of 60 CPU seconds per run. These parameters were chosen after some tuning by hand. Each experiment described in the following was repeated 25 times with different random seeds.

For the purpose of studying the influence of the pheromone information, which is used for the construction process of probabilistic beam search as well as for stochastic sampling, we performed experiments with a version of Beam-ACO in which the pheromone update was switched off. This has the effect of removing the learning mechanism from Beam-ACO. In the presentation of the results this version is denoted by **no ph**. Moreover, we wanted to study the importance of stochastic sampling. Remember that, at each step of probabilistic beam search, a number of maximally $\lfloor \mu \cdot k_{bw} \rfloor$ extensions of partial solutions are chosen. Then, based on the results of stochastic sampling, procedure Reduce removes extensions until only the best k_{bw} extensions with respect to stochastic sampling are left. In order to learn if this reduction step is important, we repeated all the experiments with a version of Beam-ACO in which $\mu = 1$ and $k_{bw} = 15$. The setting of $k_{bw} = 15$ was chosen in order to be fair with the algorithm version that uses parameter settings $\mu = 1.5$ and $k_{bw} = 10$. Note that when $\mu = 1$, procedure Reduce is never invoked and stochastic sampling is never performed. This is because with $\mu = 1$ never more than k_{bw} candidate extensions will be chosen, which makes the use of Reduce unnecessary. In the presentation of the results the corresponding version of Beam-ACO is denoted by **no ss**. Finally, we study how important stochastic sampling is as an estimate. This was done by applying it only after a certain number of construction steps of each probabilistic beam search, that is, once the partial solutions in the beam of a probabilistic beam search contain already a certain percentage of the total number of customers. More specifically, for the first $(n-(r^{s} \cdot n)/100)$ construction steps of probabilistic beam search, stochastic sampling is not used at all. Instead, Reduce simply selects k_{bw} partial solutions at random. In contrast, for the remaining $(r^{s} \cdot n)/100$ construction steps of probabilistic beam search, procedure Reduce uses the estimate provided by stochastic sampling for the elimination of partial solutions. Henceforth, we refer to parameter r^{s} as the *rate of stochastic sampling*. The value of this parameter is given as a percentage, where 0% means that no stochastic sampling is ever performed, while 100% refers to the Beam-ACO approach that always uses stochastic sampling. In our experiments we tested the following rates of stochastic sampling: $r^{s} \in \{25\%, 50\%, 75\%, 85\%, 100\%\}$. In the presentation of the results the corresponding algorithm versions are simply denoted by their value of parameter r^{s}.

Experiments without local search: Fig. 2 shows the results of the different experiments described above for five representative problem instances from the benchmark set provided by Potvin and Bengio [24]. These results were obtained without using local search. The barplots (in gray) compare the results with respect to the mean ranks obtained by each algorithm version over 25 runs. The ranks are calculated by sorting all solutions lexicographically. Moreover, the standard deviations of the ranks are shown as error bars. On the other hand, the boxplots (in white) show the distribution of computation time in seconds required by each algorithm version. Note that the notion of computation time refers to the time at which the best solution of a run was found.

The following conclusions can be drawn from Fig. 2. First, when no pheromone information is used (**no ph**), the performance of the algorithm drops significantly. Second, the use of stochastic sampling seems essential to achieve satisfactory results. When no stochastic sampling is used (**no ss**), the results achieved are worse than the ones obtained by Beam-ACO with stochastic sampling, and the algorithm requires significantly more computation time. Finally, the results of the algorithm variants using different rates of stochastic sampling show a clear pattern. The performance of the algorithm increases with increasing rate of stochastic sampling. Starting from rates of stochastic sampling of at least 75%, the performance of the algorithm is already very close to the performance of Beam-ACO when always using stochastic sampling. This result indicates that stochastic sampling helps the algorithm to converge to better solutions.

Experiments with local search: We repeated the above experiments, this time enabling local search. Fig. 3 shows the corresponding results. First, the local search that we implemented is very effective, hence, when enabled, some instances become easily solvable (e.g. rc.203.3). Moreover, the computation time needed by the algorithm enhanced with local search is in general much lower than the computation time needed by the algorithm not using local search. This is because when local search is enabled, the algorithm can reach good solutions much quicker. It is also worth to mention that pheromone information and stochastic sampling are still necessary, even when using local search. This is confirmed by the fact that, in general, algorithm performance drops significantly when disabling pheromone information (see algorithm version **no ph**) and without the use of stochastic sampling (see algorithm version **no ss**). Interestingly, the

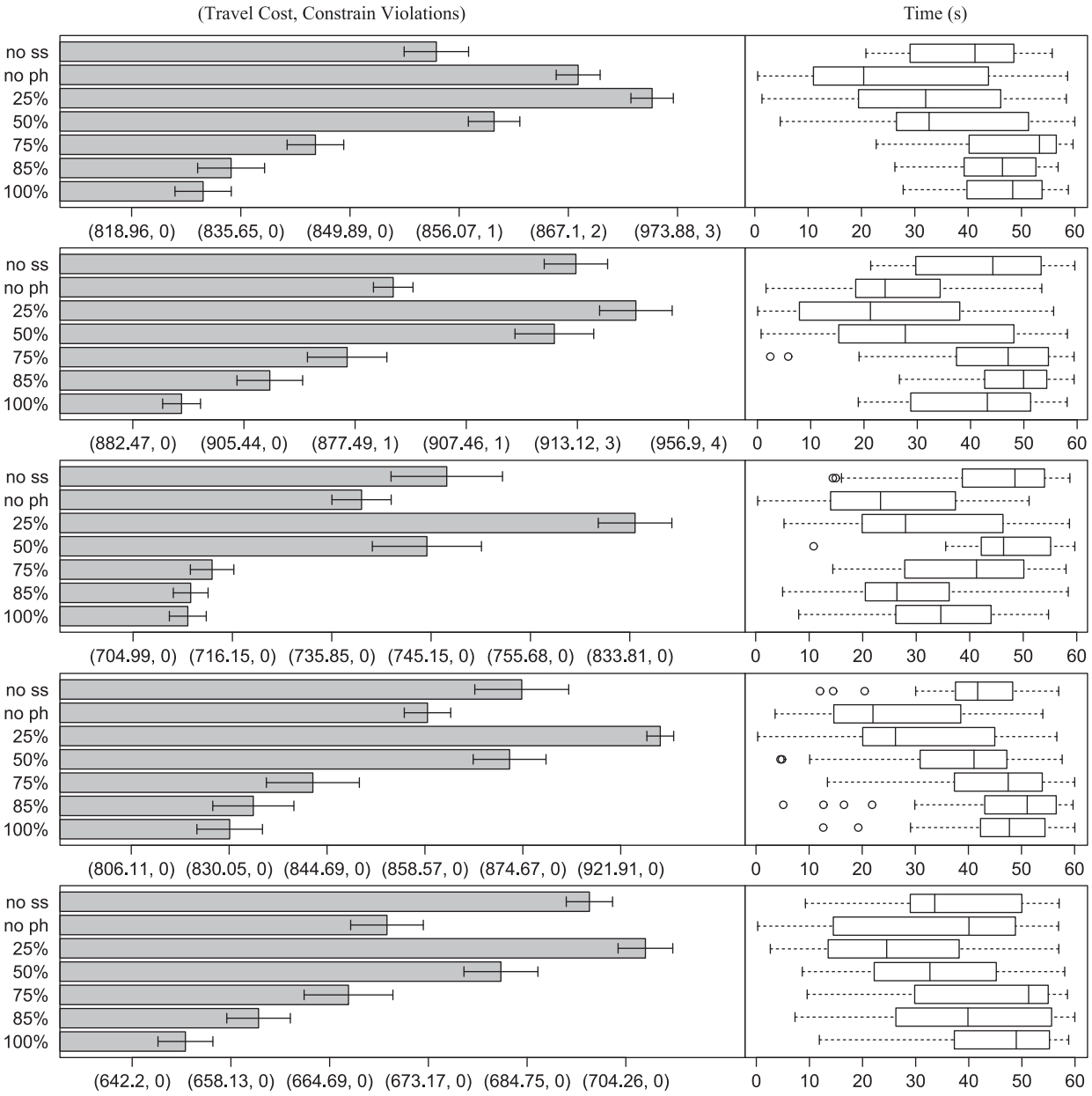

Fig. 2. Results concerning the analysis of Beam-ACO with local search disabled. From top to bottom the graphics concern instances rc.203.3, rc.204.1, rc.207.2, rc.208.1, and rc.208.3.

algorithm behaves now differently for what concerns the rate of stochastic sampling. Remember that when no local search is used, the algorithm performed better when the rate of stochastic sampling was high. In contrast, when local search was used rather low values of stochastic sampling seem to be advised. In order to avoid having to fine-tune the rate of stochastic sampling for the final experimentation we decided instead to reduce the time spent by stochastic sampling simply by reducing the number of samples taken for each partial solution. As shown in the following section, the resulting algorithm is able to achieve state-of-the-art results on most sets of benchmark instances.

4.2. Comparison to the state of the art

In the following we compare the performance of Beam-ACO with the results of the best algorithms found in the literature. For this purpose we consider seven available sets of benchmark instances:

1. The first set consists of 30 instances originally provided by Potvin and Bengio [24] and derived from Solomon's RC2 VRPTW instances [25]. These instances are very diverse in structure. The number of customers (n) ranges from 3 to 44 customers.
2. The second set of benchmark instances, by Langevin et al. [4], consists of seven instance classes of 10 instances each. Instances are grouped by number of customers and time window width.
3. The third benchmark set consists of 27 classes of five instances each. All instances were proposed and solved to optimality by Dumas et al. [5]. Instance size ranges from 20 to 200 customers.
4. Gendreau et al. [11] provided the fourth benchmark set consisting of 120 instances grouped into 26 classes with equal

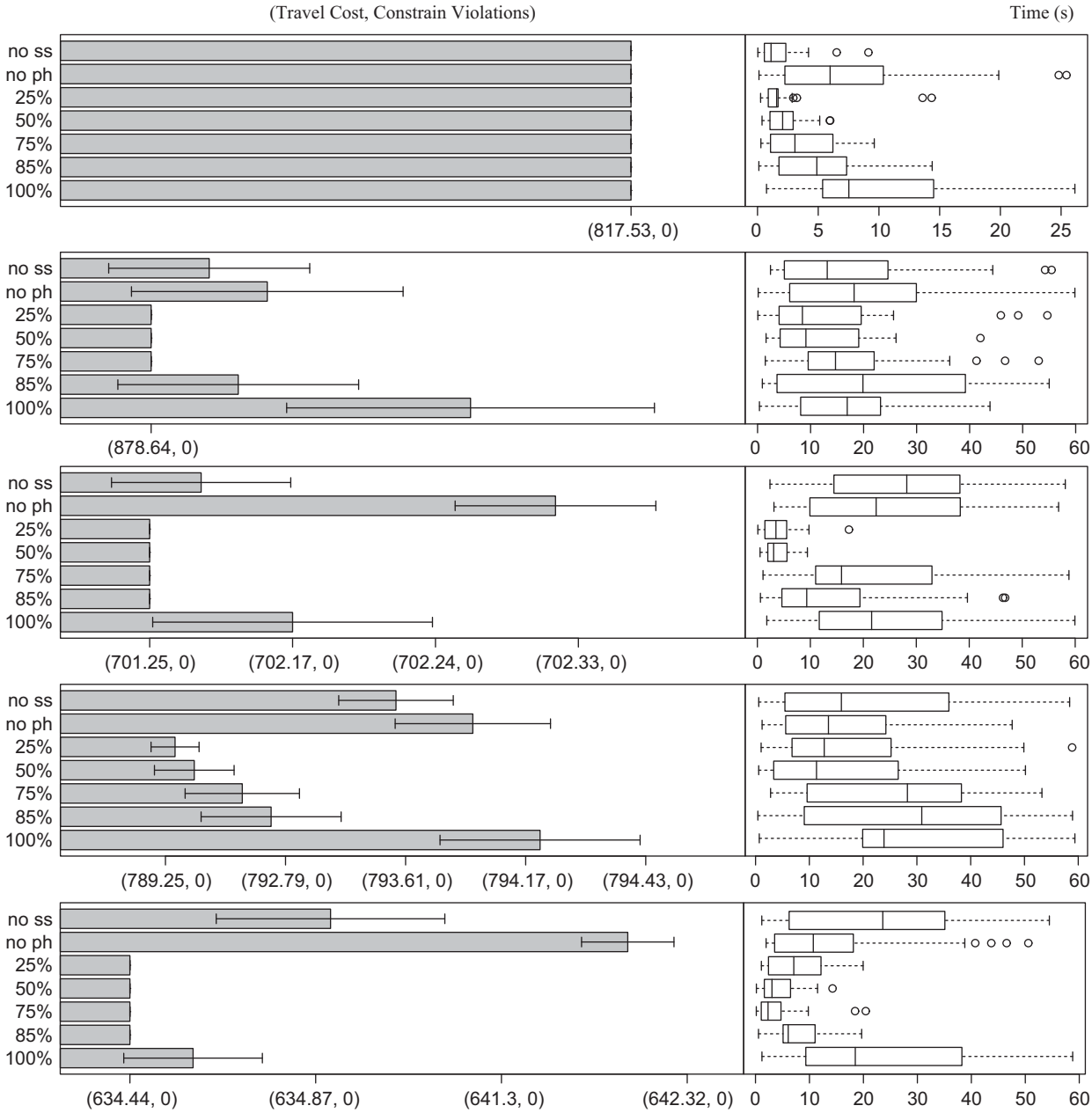

Fig. 3. Results concerning the analysis of Beam-ACO with local search enabled. From top to bottom the graphics concern instances rc.203.3, rc.204.1, rc.207.2, rc.208.1, and rc.208.3.

number of customers and time window width. These instances were obtained from the instances proposed by Dumas et al. [5] by extending the time windows by 100 units, resulting in time windows in the range from 120 to 200 time units.

5. The fifth set of benchmark instances, proposed by Ohlmann and Thomas [13], contains 25 instances grouped into five classes. The instances were derived from the instances with 150, respectively 200, customers proposed by Dumas et al. [5] by extending the time windows by 100 time units.
6. The sixth benchmark set consists of 50 asymmetric TSPTW instances introduced by Ascheuer [26]. These are real-world instances derived "from an industry project with the aim to minimize the unloaded travel time of a stacker crane within an automated storage system". They were tackled by Ascheuer et al. [6], Focacci et al. [9], and Balas and Simonetti [7].
7. Finally, the seventh benchmark set contains 27 symmetric instances proposed by Pesant et al. [8]. While they were derived from Solomon's RC2 VRPTW instances [25], they are different from the instances proposed by Potvin and Bengio [24]. These instances were also utilized by Focacci et al. [9].

We performed a set of preliminary experiments in order to find appropriate parameter settings. The goal of these preliminary experiments was to find parameter values that produce overall good results across most instances, even if they were not the optimal settings for all instances. On the basis of these experiments we chose $k_{\text{bw}} = 5$, $\mu = 1.5$, $N^{\text{s}} = 1$, $q_0 = 0.9$, and a time limit of 60 CPU seconds per run and per instance. Local search was always enabled. Results are presented in the same way as in the state-of-the-art paper by Ohlmann and Thomas [13]. In particular, we provide the relative percentage deviation (RPD), that is, $100 \cdot (\text{value} - \text{best-known})/\text{best-known}$. Since Beam-ACO is a stochastic algorithm, we provide both the mean and standard deviation (sd) of the RPD values over 15 runs with different

random seeds. We also provide the mean and standard deviation of the CPU time (T_{cpu}) required to find the best solution returned by each run of the Beam-ACO algorithm. For completeness, and following Ohlmann and Thomas [13], we also provide the CPU times reported in the original publications of each algorithm. All CPU times are rounded to seconds, since differences smaller than one second are irrelevant in this context. These CPU times, however, were obtained with very different languages and computer systems, and hence, are not directly comparable. After the presentation of the results we provide a discussion on computation time in Section 4.4.

Table 2 shows the results obtained by Beam-ACO for the instances by Potvin and Bengio [24]. The results are shown in comparison with compressed annealing (CA) [13], a dynamic programming algorithm (DP) [7], and the best results obtained by previous heuristic methods [11,12]. Although the results reported in Calvo [12] are generally better than those reported by Gendreau et al. [11], this is not true for a few instances (marked with a *b*) where Calvo's algorithm obtained higher travel cost or was not able to find a feasible solution. The fact that Beam-ACO usually obtains standard deviations of zero suggests that Beam-ACO is able to obtain the optimal solutions in all runs for most instances. By comparison, CA shows a higher variability of the results. The performance of DP is extremely good on some instances and quite bad on others. The heuristics outperform Beam-ACO on two instances, rc208.1 and rc204.1 (see the footnote marked with an exclamation mark). However, they are generally worse than Beam-ACO and CA for most of the instances. Moreover, Beam-ACO is always able to find a feasible solution, which is not the case for any of the other algorithms. In particular, for three instances, CA only finds a feasible solution in 90% of the runs.

Table 3 presents the results for the instances of Langevin et al. [4]. Following Ohlmann and Thomas [13], we average the statistics of 15 runs for each instance over the 10 instances of each instance class. Beam-ACO is compared with the known optimal solutions [12], and compressed annealing (CA) [13]. The results of the heuristic method by Calvo [12] are comparable to those obtained by CA, and, hence, they are not shown. Beam-ACO is always able to obtain the best-known solution. In fact, this set of instances seems to be easily solvable by any of the three algorithms that were applied to this benchmark set.

Table 4 shows results for the instances proposed by Dumas et al. [5]. The statistics of 15 applications to each instance are averaged over the five instances of each instance class. We compare Beam-ACO with results obtained by an exact method [5], compressed annealing (CA) [13], and the best value achieved among the following three algorithms (Heuristic): Calvo [12], Gendreau et al. [11], and the tabu search of Carlton and Barnes [10]. In terms of quality, Beam-ACO shows the highest robustness,

Table 2
Results for instances from Potvin and Bengio [24].

Instance	n	Best known	Beam-ACO				CA [13]			DP [7]		Heuristic	
			Mean RPD	Sd. RPD	Mean T_{cpu}	Sd. T_{cpu}	Mean RPD	Sd. RPD	Mean T_{cpu}	RPD	T_{cpu}	RPD	T_{cpu}
rc201.1	20	444.54	0.00	0.00	0	0	0.00	0.00	5	0.00[a]	2	0.00[b]	0
rc201.2	26	711.54	0.00	0.00	0	0	0.00	0.00	6	0.00[a]	3	0.00[b]	0
rc201.3	32	790.61	0.00	0.00	2	3	0.00	0.00	9	0.00[a]	4	0.00[b]	3
rc201.4	26	793.64	0.00	0.00	0	0	0.00(90%)	0.00	6	0.00[a]	3	0.00[b]	0
rc202.1	33	771.78	0.00	0.00	0	0	0.05	0.02	11	0.07	223	0.05[b]	8
rc202.2	14	304.14	0.00	0.00	0	0	0.00	0.00	5	0.00[a]	2	0.00[b]	0
rc202.3	29	837.72	0.00	0.00	1	1	0.00	0.00	7	0.00[a]	45	0.22[b]	0
rc202.4	28	793.03	0.00	0.00	0	0	0.00	0.00	9	0.78	212	0.00[b]	2
rc203.1	19	453.48	0.00	0.00	0	0	0.00	0.00	7	0.00[a]	15	0.00[b]	0
rc203.2	33	784.16	0.00	0.00	0	0	0.00	0.00	11	3.14	404	0.00[b]	4
rc203.3	37	817.53	0.00	0.00	2	2	0.03	0.11	12	*infeasible*		0.23[b]	14
rc203.4	15	314.29	0.00	0.00	0	0	0.00	0.00	5	0.00[a]	3	0.00[b]	0
rc204.1	46	868.76(!)	1.14	0.00	11	10	1.34(90%)	0.35	14	*infeasible*		0.00[b](!)	35
rc204.2	33	662.16	0.00	0.00	8	7	0.71	1.29	10	0.00	77	0.57[b]	8
rc204.3	24	455.03	0.00	0.00	0	0	0.96	0.50	9	2.46	639	0.00[b]	4
rc205.1	14	343.21	0.00	0.00	0	0	0.00	0.00	4	0.00[a]	2	0.00[b]	0
rc205.2	27	755.93	0.00	0.00	0	0	0.00(90%)	0.00	7	0.00[a]	5	0.00[b]	0
rc205.3	35	825.06	0.00	0.00	1	1	0.00	0.00	10	0.00	42	0.00[b]	21
rc205.4	28	760.47	0.00	0.00	5	5	0.00	0.00	7	0.00[a]	5	0.26[c]	6
rc206.1	4	117.85	0.00	0.00	0	0	0.00	0.00	3	0.00[a]	0	0.00[b]	0
rc206.2	37	828.06	0.00	0.00	0	0	0.01	0.04	11	0.00	33	1.70[c]	33
rc206.3	25	574.42	0.00	0.00	1	1	0.00	0.00	9	0.00	38	0.00[b]	0
rc206.4	38	831.67	0.00	0.00	3	2	0.10	0.24	11	0.00	46	0.71[b]	8
rc207.1	34	732.68	0.00	0.00	0	0	0.00	0.00	11	0.43	70	0.07[b]	4
rc207.2	31	701.25	0.00	0.00	7	5	0.00	0.00	10	0.00	61	2.40[c]	16
rc207.3	33	682.40	0.00	0.00	1	1	0.00	0.00	11	2.28	1128	0.29[c]	17
rc207.4	6	119.64	0.00	0.00	0	0	0.00	0.00	3	0.00[a]	0	0.00[b]	0
rc208.1	38	789.25	0.30	0.29	19	21	0.58	0.36	12	0.55	1141	0.00[b]	10
rc208.2	29	533.78	0.00	0.00	1	1	0.17	0.54	10	0.00	59	0.67[b]	2
rc208.3	36	634.44	0.00	0.00	12	11	0.95	0.84	11	3.32	122	2.31[b]	8

(!) Note: Even after thorough testing the best result achieved by Beam-ACO for instance rc204.1 was 878.64. In contrast, the best value reported by Calvo [12] is 868.76. Calvo [12] states that the instance has 44 customers, when in fact it has 46 costumers.

[a] Optimal value.
[b] Heuristic solution obtained by Calvo [12].
[c] Heuristic solution obtained by Gendreau et al. [11].

Table 3
Results for instances proposed by Langevin et al. [4].

Data set		Best known	T_{cpu}	Beam-ACO				CA [13]		
n	Time window width			Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Mean Sd. T_{cpu}	Mean RPD	Mean Sd. RPD	Mean T_{cpu}
20	30	724.7[a]	0	0.00	0.00	0	0	0.00	0.00	5
	40	721.5[a]	1	0.00	0.00	0	0	0.00	0.00	5
40	20	982.7[a]	2	0.00	0.00	0	0	0.00	0.00	7
	40	951.8[a]	7	0.00	0.00	0	0	0.00	0.00	7
60	20	1215.7	–	0.00	0.00	0	1	0.00	0.00	9
	30	1183.2	–	0.00	0.00	0	0	0.00	0.00	12
	40	1160.7	–	0.00	0.00	3	2	0.00	0.01	14

[a] Optimal solution [12].

Table 4
Results for instances proposed by Dumas et al. [5].

Data set		Exact [5]		Beam-ACO				CA [13]			Heuristic [12]	
n	Time window width	Optimal value	T_{cpu}	Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Mean Sd. T_{cpu}	Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Mean RPD	Mean T_{cpu}
20	20	361.2	0	0.00	0.00	0	0	0.00	0.00	5	0.00	0
	40	316.0	0	0.00	0.00	0	0	0.00	0.00	5	0.00	0
	60	309.8	0	0.00	0.00	0	0	0.00	0.00	5	0.00	0
	80	311.0	0	0.00	0.00	0	0	0.00(98%)	0.00	5	0.00	0
	100	275.2	1	0.00	0.00	0	0	0.00	0.00	6	0.00	0
40	20	486.6	0	0.00	0.00	0	0	0.00	0.00	7	0.00	3
	40	461.0	0	0.00	0.00	0	0	0.00	0.00	10	0.00	3
	60	416.4	4	0.00	0.00	1	0	0.00	0.02	12	0.00	5
	80	399.8	8	0.00	0.00	1	1	0.05	0.25	12	0.00	5
	100	377.0	31	0.00	0.00	4	4	0.11	0.27	12	0.00	6
60	20	581.6	0	0.00	0.00	0	1	0.00	0.03	13	0.00	8
	40	590.2	1	0.00	0.00	1	1	0.12	0.41	16	0.00[a]	37
	60	560.0	7	0.00	0.02	5	5	0.04	0.12	16	0.00	11
	80	508.0	47	0.00	0.02	6	6	0.24(98%)	0.39	16	0.20	18
	100	514.8	200	0.16	0.19	16	12	0.33	0.37	16	0.31	26
80	20	676.6	0	0.00	0.00	2	3	0.03	0.24	20	0.00	43
	40	630.0	3	0.00	0.00	2	9	0.02	0.03	21	0.00	69
	60	606.4	55	0.12	0.10	18	12	0.13(98%)	0.26	21	1.72[b]	89
	80	593.8	220	0.13	0.17	21	14	0.29(98%)	0.29	21	0.10	60
100	20	757.6	103	0.00	0.01	9	9	0.03	0.11	24	0.00[a]	175
	40	701.8	129	0.03	0.07	14	12	0.06(86%)	0.14	25	0.14[b]	1
	60	696.6	148	0.01	0.03	17	13	0.17(94%)	0.43	25	0.00	148
150	20	868.4	2	0.05	0.06	20	16	0.12	0.21	36	0.02	420
	40	834.8	116	0.06	0.06	17	13	0.11	0.26	36	0.22[b]	5
	60	805.0	463	2.09	0.21	29	18	2.10(84%)	0.60	37	1.91	630
200	20	1009.0	7	0.05	0.03	80	61	0.13(98%)	0.24	50	0.10	1456
	40	984.2	251	0.08	0.06	115	80	0.25(98%)	0.17	50	0.12	2106

Results for $n = 200$ have a time limit of 300 seconds.

[a] The best-known heuristic solution value is found by Gendreau et al. [11].
[b] The best-known heuristic solution value is obtained by Carlton and Barnes [10].

achieving a feasible solution in all runs. Moreover, Beam-ACO both finds the optimal solution and has a low variability in most of the runs. In comparison, compressed annealing does not find a feasible solution in all of the runs, e.g. in only 84% of the runs performed for the instances with $n = 150$ and time window 60. On the other hand, both Beam-ACO and compressed annealing typically match or outperform the best-known heuristic values.

Table 5 compares the results of Beam-ACO with compressed annealing (CA) [13] and the heuristic of Calvo [12] for the instances proposed by Gendreau et al. [11]. These instances have wide time windows, which means that available exact algorithms have problems obtaining feasible solutions in a reasonable computation time. The results of Gendreau et al. [11] for their own instances are always worse and generally obtained in more computation time than those reported by Calvo [12], and, hence, they are not shown. In general, Beam-ACO obtains better results than compressed annealing in terms of robustness, average quality and low variability. In fact, for the instances with 60 customers and a time window width of 140, compressed annealing finds a feasible solution in only 92% of the runs,

Table 5
Results for instances proposed by Gendreau et al. [11].

Data set		Best known value	Beam-ACO				CA [13]			Calvo [12]	
n	Time window width		Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Sd. T_{cpu}	Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Mean RPD	Mean T_{cpu}
20	120	265.6	0.00	0.00	3	2	0.00	0.00	7	0.60	0
	140	232.8	0.13	0.17	4	4	0.00	0.00	8	11.51	0
	160	218.2	0.00	0.00	1	1	0.00	0.00	8	19.16	0
	180	236.6	0.00	0.00	1	1	0.00	0.00	8	3.38	0
	200	241.0	0.00	0.00	0	0	0.00	0.00	8	0.83	0
40	120	360.0	4.94	0.00	3	4	5.14	0.61	12	0.00	5
	140	348.4	4.79	0.11	9	6	4.74	0.26	12	0.00	9
	160	326.8	0.03	0.03	3	4	0.03	0.06	12	3.18	10
	180	326.8	1.56	0.49	14	13	2.17	0.70	12	0.00	12
	200	313.8	0.29	0.06	7	8	0.35	0.32	12	0.45	16
60	120	451.0	0.07	0.04	18	11	0.51	0.71	16	7.18	30
	140	452.0[a]	0.18	0.07	10	8	0.49(92%)	0.51	16	0.53	28
	160	448.6	3.63	0.04	11	9	3.72	0.45	16	0.00	34
	180	421.2	0.33	0.28	1	14	0.85	1.28	16	2.75	41
	200	427.4[a]	0.12	0.21	22	16	0.70	0.75	16	0.14	57
80	100	578.8[a]	0.40	0.26	22	16	–	–	–	0.24	72
	120	541.4	0.65	0.24	17	12	0.42	0.52	20	1.55	64
	140	506.8[a]	0.75	0.51	26	17	1.16	1.10	20	3.71	75
	160	502.8	0.91	0.30	24	16	2.09	1.07	21	0.00	82
	180	489.0	3.19	0.29	24	14	3.27	0.84	21	0.00	116
	200	482.6	0.87	0.48	28	17	0.62	0.85	20	0.29	158
100	80	666.4	0.20	0.20	21	18	0.38	0.45	25	0.24	193
	100	642.0	0.65	0.39	24	18	0.45	0.50	24	0.31	119
	120	599.4[a]	0.60	0.37	22	16	0.77	0.55	24	2.50	167
	140	550.2[a]	0.62	0.42	25	18	5.69	0.53	24	7.49	201
	160	556.6[a]	0.93	0.56	31	18	5.91	0.65	24	2.48	214

[a] New best-known solution found by Beam-ACO.

Table 6
Results for instances proposed by Ohlmann and Thomas [13].

Data set		Best known value	Beam-ACO ($T_{cpu} \leq 60$)				Beam-ACO ($T_{cpu} \leq 300$)				CA [13]		
n	Time window width		Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Sd. T_{cpu}	Mean RPD	Mean Sd. RPD	Mean T_{cpu}	Sd. T_{cpu}	Mean RPD	Mean Sd. RPD	Mean T_{cpu}
150	120	724.0	0.80	0.39	26	17	0.47	0.22	118	86	0.98	0.87	36
	140	697.2[a]	1.64	0.62	32	16	0.85	0.46	132	77	1.15	0.82	36
	160	672.6[a]	1.16	0.65	32	14	0.54	0.28	144	83	1.38	0.85	36
200	120	806.4[a]	1.50	0.69	37	12	0.55	0.33	144	73	1.50	0.93	50
	140	802.4[a]	1.48	0.66	40	10	0.69	0.44	166	73	1.31	0.81	49

[a] New best-known solution found by Beam-ACO.

whereas Beam-ACO always finds a feasible solution. Although the heuristic of Calvo [12] obtains the best known results for a few instances, the average result of Beam-ACO is significantly better for the remaining ones, being able to find new best-known solutions in seven cases.

Table 6 examines the performance of Beam-ACO on the instances proposed by Ohlmann and Thomas [13]. The only algorithm ever applied to these instances is compressed annealing from the same paper. These instances should be particularly difficult for both heuristic and exact methods, since they involve a large number of customers and wide time windows. For the usual time limit of 60 seconds, the results of Beam-ACO are already comparable to those of CA, slightly better for some instances and slightly worse for others. For this instance set we additionally applied Beam-ACO with a time limit of 300 seconds to each instance. This was done in an attempt to assess if Beam-ACO was able to further improve when given more computation time. The results show that this is indeed the case. Beam-ACO achieves a significantly lower travel cost and lower variability than CA.

Table 7 concerns the results obtained for the asymmetric instances proposed by Ascheuer [26]. Results are given for Beam-ACO, dynamic programming (DP) [7], the branch-and-cut algorithm (B&C) of Ascheuer et al. [6], and the hybrid exact algorithm (Hybrid) of Focacci et al. [9]. The latter combines constraint programming with optimization techniques based on solving a relaxation of the original problem. They propose two variants: one based on the assignment problem relaxation (AP-bound), and another that also incorporates a Lagrangean relaxation (Lagrangean-bound). Neither variant consistently outperforms the other, and hence, for comparison with Beam-ACO, we use the best result obtained by either of them. The quality of the results of Beam-ACO is very good up to 150 customers, finding the best-known solution in most of the runs. For higher number of customers, the results are still within 1% of

Table 7
Results for asymmetric instances proposed by Ascheuer [26].

Instance	n	Best value	Beam-ACO				DP [7]		B&C [6]		Hybrid [9]	
			Mean RPD	Sd. RPD	Mean T_{cpu}	Sd. T_{cpu}	RPD	T_{cpu}	RPD	T_{cpu}	RPD	T_{cpu}
rbg010a	12	671	0.00	0.00	0	0	0.00[a]	0	0.00[a]	0	0.00[a,b]	0
rbg016a	18	938	0.00	0.00	0	0	0.00[a]	1	0.00[a]	0	0.00[a,b]	0
rbg016b	18	1304	0.00	0.00	0	0	0.00[a]	2	0.00[a]	9	0.00[a,b]	0
rbg017.2	17	852	0.00	0.00	0	0	0.00[a]	5	0.00[a]	0	0.00[a,b]	0
rbg017	17	893	0.00	0.00	1	1	0.00[a]	2	0.00[a]	1	0.00[a,b]	0
rbg017a	19	4296	0.00	0.00	0	0	0.00[a]	3	0.00[a]	0	0.00[a,b]	0
rbg019a	21	1262	0.00	0.00	0	0	0.00[a]	2	0.00[a]	0	0.00[a,b]	0
rbg019b	21	1866	0.00	0.00	0	0	0.00[a]	3	0.00[a]	55	0.00[a,b]	0
rbg019c	21	4536	0.00	0.00	0	0	0.00[a]	8	0.00[a]	9	0.00[a,b]	0
rbg019d	21	1356	0.00	0.00	0	0	0.00[a]	2	0.00[a]	1	0.00[a,b]	0
rbg020a	22	4689	0.00	0.00	0	0	0.00[a]	8	0.00[a]	0	0.00[a,b]	0
rbg021.2	21	4528	0.00	0.00	2	2	0.00[a]	11	0.00[a]	0	0.00[a,b]	0
rbg021.3	21	4528	0.00	0.00	9	8	0.00[a]	11	0.00[a]	27	0.00[a,b]	0
rbg021.4	21	4525	0.00	0.00	0	0	0.00[a]	29	0.00[a]	6	0.00[a,b]	0
rbg021.5	21	4515	0.02	0.02	13	19	0.00[a]	76	0.00[a]	7	0.00[a,b]	0
rbg021.6	21	4480	0.00	0.00	8	6	0.00[a]	92	0.00[a]	1	0.00[a,b]	1
rbg021.7	21	4479	0.00	0.00	2	2	0.00	224	0.00[a]	4	0.00[a,b]	1
rbg021.8	21	4478	0.00	0.00	1	1	0.00	267	0.00[a]	17	0.00[a,b]	1
rbg021.9	21	4478	0.00	0.00	1	1	0.00	285	0.00[a]	26	0.00[a,b]	1
rbg021	21	4536	0.00	0.00	0	0	0.00[a]	8	0.00[a]	8	0.00[a,b]	0
rbg027a	29	5091	0.00	0.00	0	0	0.00	11	0.00[a]	2	0.00[a,b]	0
rbg031a	33	1863	0.00	0.00	1	1	0.00[a]	7	0.00[a]	2	0.00[a,c]	3
rbg033a	35	2069	0.00	0.00	0	0	0.00[a]	5	0.00[a]	2	0.00[a,b]	1
rbg034a	36	2220	0.09	0.00	2	2	0.00[a]	11	0.09[a]	1	0.09[a,b]	55
rbg035a.2	37	2056	0.04	0.02	15	17	0.15	650	0.00[a]	2	0.00[a,b]	37
rbg035a	37	2144	0.00	0.00	1	1	0.00[a]	7	0.00[a]	65	0.00[a,b]	4
rbg038a	40	2480	0.00	0.00	6	8	0.00[a]	8	0.00[a]	4232	0.00[cb]	0
rbg040a	42	2378	0.02	0.03	15	16	0.00[a]	13	0.00[a]	752	0.00[cb]	738
rbg041a	43	2598	0.06	0.06	34	15	0.00[a]	15	0.58	5h	0.04[b]	1800
rbg042a	44	2772	0.16	0.07	24	16	0.00[a]	61	0.87	5h	0.00[a,b]	150
rbg048a	50	9387	0.11	0.05	26	16	*infeasible*		0.38	5h	0.01[b]	1800
rbg049a	51	10 019	0.05	0.04	26	17	0.01	281	0.16	5h	0.03[b]	1800
rbg050a	52	2953	0.30	0.04	20	15	0.58	1123	0.00[a]	19	0.00[ac]	96
rbg050b	52	9863	0.05	0.04	28	15	0.06	360	0.30	5h	0.15[c]	1800
rbg050c	52	10 026	0.07	0.04	40	17	*infeasible*		0.08	5h	0.15[b]	1800
rbg055a	57	3761	0.00	0.00	11	14	0.00[a]	16	0.00[a]	6	0.00[a,b]	2
rbg067a	69	4625	0.00	0.02	15	13	0.00[a]	18	0.00[a]	6	0.00[a,b]	4
rbg086a	88	8400	0.06	0.05	24	19	0.00[a]	18	0.01	5h	0.42[c]	1800
rbg092a	94	7158	0.05	0.03	18	15	0.00[a]	30	0.25	5h	0.22[c]	1800
rbg125a	127	7936	0.05	0.04	32	19	0.00[a]	31	0.01[a]	230	0.47[c]	1800
rbg132.2	134	8191	0.45	0.14	38	17	0.00	1135	0.51	5h	–	–
rbg132	134	8468	0.19	0.08	27	16	0.00[a]	39	0.47	5h	–	–
rbg152.3	154	9791	0.15	0.06	35	15	0.00	2765	0.53	5h	–	–
rbg152	154	10 032	0.06	0.03	25	18	0.00[a]	37	0.09	5h	–	–
rbg172a	174	10 950	0.39	0.16	35	17	0.00	812	0.89	5h	–	–
rbg193.2	195	12 143	0.51	0.10	37	16	0.00	2138	0.58	5h	–	–
rbg193	195	12 535	0.29	0.14	37	15	0.00	807	0.30	5h	–	–
rbg201a	203	12 948	0.48	0.12	37	14	0.00	809	0.83	5h	–	–
rbg233.2	235	14 496	0.61	0.10	43	11	0.00	2505	0.77	5h	–	–
rbg233	235	14 992	0.56	0.15	42	10	0.00	975	0.65	5h	–	–

[a] Reported as optimal (there are small differences between the optimal values reported by Balas and Simonetti [7] and Focacci et al. [9]).
[b] AP-bound [9].
[c] Lagrangean-bound [9].

the best-known solutions. Furthermore, Beam-ACO is able to find a feasible solution for all instances and in all runs, whereas DP fails to find a feasible solution for two instances. In general, both DP and Hybrid outperform B&C, however, neither DP nor Hybrid are consistently better than B&C for all instances. For some instances they find the optimal solution very fast (e.g. DP on rbg152), whereas for other instances they require a long computation time (e.g. Hybrid on rbg040a). Sometimes their results are worse than those obtained by Beam-ACO even after very long running times (e.g. rbg050b and rbg050c). For large instances ($n > 150$), Beam-ACO is able to obtain good approximations (less than 1% deviation) to the results obtained by the exact algorithms within a much shorter computation time.

Table 8 provides the results for the symmetric instances proposed by Pesant et al. [8]. For this benchmark set, previous results are available from an exact algorithm based on constraint programming by Pesant et al. [8] and from two variants of the hybrid algorithm (Hybrid) by Focacci et al. [9]. In general, Hybrid completely outperforms the results of Pesant et al. [8]. However, when Hybrid fails to find an optimal solution, the best-known solution is always found by Pesant et al. [8], who let their algorithm run for a whole day. For this benchmark set, Beam-ACO is not only able to find the optimal (or best-known) solution in all runs for almost all instances but also requires significantly less time than the exact algorithms, even considering hardware differences.

Table 8
Results for symmetric instances proposed by Pesant et al. [8].

Instance	n	Best value	Beam-ACO				Previous results	
			Mean RPD	Sd. RPD	Mean T_{cpu}	Sd. T_{cpu}	RPD	T_{cpu}
rc201.0	25	628.62	0.00	0.00	0	0	0.00[a]	0
rc201.1	28	654.70	0.00	0.00	0	0	0.00[a]	2
rc201.2	28	707.65	0.00	0.00	0	0	0.00[b]	0
rc201.3	19	422.54	0.00	0.00	0	0	0.00[a]	0
rc202.0	25	496.22	0.00	0.00	0	0	0.00[b]	1
rc202.1	22	426.53	0.00	0.00	0	0	0.00[a]	4
rc202.2	27	611.77	0.00	0.00	0	0	0.00[a]	3
rc202.3	26	627.85	0.00	0.00	0	0	0.00[a]	33
rc203.0	35	727.45	0.00	0.00	1	0	1.01[c]	1 day
rc203.1	37	726.99	0.00	0.00	3	3	0.04[c]	1 day
rc203.2	28	617.46	0.00	0.00	1	1	0.00[b]	94
rc204.0	32	541.45	0.00	0.00	0	0	0.00[b]	353
rc204.1	28	485.37	0.00	0.00	2	2	0.00[b]	3
rc204.2	40	778.40	0.00	0.01	19	14	0.08[c]	1 day
rc205.0	26	511.65	0.00	0.00	0	0	0.00[b]	8
rc205.1	22	491.22	0.00	0.00	0	0	0.00[a]	0
rc205.2	28	714.69	0.00	0.00	1	1	0.00[a]	1289
rc205.3	24	601.24	0.00	0.00	0	0	0.00[a]	5
rc206.0	35	835.23	0.00	0.00	5	5	0.00[b]	338
rc206.1	33	664.73	0.00	0.00	3	3	0.00[b]	23
rc206.2	32	655.37	0.00	0.00	2	2	0.00[b]	24
rc207.0	37	806.69	0.00	0.00	0	0	0.00[a]	572
rc207.1	33	726.36	0.00	0.00	2	2	0.00[b]	322
rc207.2	30	546.41	0.00	0.00	0	0	0.00[b]	15
rc208.0	44	820.56	0.00	0.00	7	8	0.07[c]	1 day
rc208.1	27	509.04	0.00	0.00	2	2	0.00[b]	34
rc208.2	29	503.92	0.00	0.00	1	1	0.00[b]	1

[a] Optimal solution found by Hybrid [9] (AP-bound).
[b] Optimal solution found by Hybrid [9] (Lagrangean-bound).
[c] Best solution found by Pesant et al. [8].

4.3. Summary of the results

Summarizing the results above, Beam-ACO is able to obtain optimal or near-optimal (less than 1% deviation) solutions in a reasonable computation time for most benchmark instances available in the literature. In comparison with compressed annealing (CA), Beam-ACO performs slightly better in terms of quality, finding feasible solutions in all runs and new best-known solutions in a few cases. In comparison with the exact algorithms, Beam-ACO is a good alternative when the goal is to obtain a good approximation in a very short time. Beam-ACO is particularly good at finding feasible solutions across a wide range of different instances.

The Beam-ACO algorithm discussed in this paper is hybridized with an efficient local search, similar to the one used by other algorithms for the TSPTW [10,13], whereas our previous work on Beam-ACO [21,22] did not make use of any local search method and was limited to one benchmark set. Therefore, we ran Beam-ACO without local search for the seven benchmark sets studied in this paper. Table 9 compares the mean RPD obtained by Beam-ACO with and without local search for each benchmark set. The comparison shows that Beam-ACO with local search always achieves the lowest mean RPD and the differences with respect to Beam-ACO without local search are large in most cases. In fact, Beam-ACO without local search was not able to find a feasible solution in some runs, whereas Beam-ACO with local search found a feasible solution in every run. These results confirm the conclusions obtained from the experiments presented in Section 4.1, which showed the benefit of adding the local search method to Beam-ACO for the TSPTW.

Table 9
Mean RPD obtained by Beam-ACO with and without local search.

Benchmark set	Mean RPD	
	With LS	Without LS
Potvin and Bengio [24] (Table 2)	0.05	1.00
Langevin et al. [4] (Table 3)	0.00	0.17
Dumas et al. [5] (Table 4)	0.10	4.30
Gendreau et al. [11] (Table 5)	0.99	9.48
Ohlmann and Thomas [13] (Table 6, $T_{cpu} \leq 60$)	1.32	29.71
Ohlmann and Thomas [13] (Table 6, $T_{cpu} \leq 300$)	0.62	27.06
Ascheuer [26] (Table 7)	0.05	2.15
Pesant et al. [8] (Table 8)	0.00	1.49

4.4. Comments on computation time

The algorithms compared in this paper were implemented in different programming languages and executed on very different computer systems. For example, CA was coded in C++ and tested on an unspecified AMD processor with 1.8 GHz and unknown cache size [13], whereas the heuristic from Calvo [12] was implemented in FORTRAN and executed on an Intel 486 CPU with 66 MHz. Therefore, it is very hard to make definitive conclusions about the relative speed of the algorithms. Nevertheless, in many cases the reported computation times provide valuable information.

The comparison of Beam-ACO with CA is difficult, because both algorithms were executed on modern processors and obtain similar computation time. Comparing processor speeds, Beam-ACO is probably slower, especially for the large instances ($n > 150$) of Table 6. However, the results obtained in Section 4.1 suggest that reducing the rate of stochastic sampling in Beam-ACO to 50% may significantly speed up the algorithm and improve the quality of the results. Hence, there is still room for improvement. Both Beam-ACO and CA are probably faster than DP for some instance classes, according to the very large

computation times required by DP for some cases in Tables 2 and 7. From the large differences observed in Table 4, we can conclude that the exact algorithm from 1995 [5] is faster than all the other approximate algorithms for instances with small time window width (less than 100). In fact, this exact algorithm is probably state-of-the-art for these classes of instances. However, when instances with a large time window width (100 or more) are concerned the exact algorithm is probably slower than both Beam-ACO and CA. As observed by Ohlmann and Thomas [13], the heuristics of Calvo [12] and Gendreau [11] would be very fast when executed on a modern processor, however, they do not consistently obtain feasible solutions, and the quality of the feasible solutions obtained by them is typically worse than the solutions obtained by Beam-ACO. Finally, B&C [7], the hybrid algorithm by Focacci et al. [9], and the exact algorithm based on constraint programming by Pesant et al. [8] are not able to outperform Beam-ACO for some instances even after such extremely long computation times that the differences in processor speed are irrelevant.

5. Conclusions

In this paper, we have proposed a Beam-ACO approach for the TSPTW for minimizing the travel cost. Beam-ACO is a hybrid between ant colony optimization and beam search that, in general, relies heavily on bounding information that is accurate and computationally inexpensive. We studied a version of Beam-ACO in which the bounding information is replaced by stochastic sampling. We also incorporated an effective local search procedure to further improve the results.

We performed experiments to study the contribution of each component of Beam-ACO, with and without local search. Our results confirmed that the use of pheromone information and stochastic sampling are needed for achieving a good performance, even when a very effective local search is applied. In addition, we carried out an extensive comparison comprising seven different benchmark sets and including the best-known exact and heuristic algorithms from the literature. The results showed that Beam-ACO achieves, in general, better results than the existing heuristic methods and is able to find good approximations in much shorter time than exact methods. Moreover, Beam-ACO is better at finding (good) feasible solutions than any of the methods reviewed. Hence, our assessment is that the proposed Beam-ACO can be seen as a state-of-the-art algorithm for the TSPTW when considering travel-cost optimization. In the future, we plan to extend this work to tackle the objective of makespan minimization, which has received less attention from the community.

References

[1] Savelsbergh MWP. Local search in routing problems with time windows. Annals of Operations Research 1985;4(1):285–305.

[2] Christofides N, Mingozzi A, Toth P. State-space relaxation procedures for the computation of bounds to routing problems. Networks 1981;11(2):145–64.

[3] Baker EK. An exact algorithm for the time-constrained traveling salesman problem. Operations Research 1983;31(5):938–45.

[4] Langevin A, Desrochers M, Desrosiers J, Gélinas S, Soumis F. A two-commodity flow formulation for the traveling salesman and makespan problems with time windows. Networks 1993;23(7):631–40.

[5] Dumas Y, Desrosiers J, Gelinas E, Solomon MM. An optimal algorithm for the traveling salesman problem with time windows. Operations Research 1995;43(2):367–71.

[6] Ascheuer N, Fischetti M, Grötschel M. Solving asymmetric travelling salesman problem with time windows by branch-and-cut. Mathematical Programming 2001;90:475–506.

[7] Balas E, Simonetti N. Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study. INFORMS Journal on Computing 2001;13(1):56–75.

[8] Pesant G, Gendreau M, Potvin J-Y, Rousseau J-M. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. Transportation Science 1998;32:12–29.

[9] Focacci F, Lodi A, Milano M. A hybrid exact algorithm for the TSPTW. INFORMS Journal on Computing 2002;14:403–17.

[10] Carlton WB, Barnes JW. Solving the traveling-salesman problem with time windows using tabu search. IIE Transactions 1996;28:617–29.

[11] Gendreau M, Hertz A, Laporte G, Stan M. A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research 1998;46:330–5.

[12] Calvo RW. A new heuristic for the traveling salesman problem with time windows. Transportation Science 2000;34(1):113–24.

[13] Ohlmann JW, Thomas BW. A compressed-annealing heuristic for the traveling salesman problem with time windows. INFORMS Journal on Computing 2007;19(1):80–90.

[14] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–80.

[15] Blum C. Beam — ACO-hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers & Operations Research 2005;32(6):1565–91.

[16] Blum C. Beam-ACO for simple assembly line balancing. INFORMS Journal on Computing 2008;20(4):618–27.

[17] Dorigo M, Stützle T. Ant colony optimization. Cambridge, MA: MIT Press; 2004.

[18] Ow PS, Morton TE. Filtered beam search in scheduling. International Journal of Production Research 1988;26:297–307.

[19] Juillé H, Pollack JB. A sampling-based heuristic for tree search applied to grammar induction. In: Proceedings of AAAI 1998—fifteenth national conference on artificial intelligence. Cambridge, MA: MIT Press; 1998. p. 776–83.

[20] Ruml W. Incomplete tree search using adaptive probing. In: Proceedings of IJCAI 2001—seventeenth international joint conference on artificial intelligence. New York: IEEE Press; 2001, p. 235–41.

[21] López-Ibáñez M, Blum, C. Beam-ACO based on stochastic sampling: a case study on the TSP with time windows. In: Proceedings of LION 3—3rd international conference on learning and intelligent optimization. Lecture notes in computer science, vol. 5851. Berlin, Germany: Springer; 2009, p. 59–73.

[22] López-Ibáñez M, Blum C, Thiruvady C, Ernst AT, Meyer B. Beam-ACO based on stochastic sampling for makespan optimization concerning the TSP with time windows. In: Proceedings of EvoCOP 2009—9th European conference evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol. 5482. Berlin, Germany: Springer; 2009. p. 97–108.

[23] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man and Cybernetics—Part B 2004;34(2):1161–72.

[24] Potvin J-Y, Bengio S. The vehicle routing problem with time windows part II: genetic search. INFORMS Journal on Computing 1996;8:165–72.

[25] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time windows. Operations Research 1987;35:254–65.

[26] Ascheuer N. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. PhD thesis, Technische Universität Berlin, Berlin, Germany; 1995.