Beam-ACO Based on Stochastic Sampling for Makespan Optimization Concerning the TSP with Time Windows*

Manuel López-Ibáñez[1], Christian Blum[1], Dhananjay Thiruvady[2,3], Andreas T. Ernst[3], and Bernd Meyer[2]

[1] ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain
{m.lopez-ibanez,cblum}@lsi.upc.edu
[2] Calyton School of Information Technology, Monash University, Australia
{dhananjay.thiruvady,bernd.meyer}@infotech.monash.edu.au
[3] CSIRO Mathematics and Information Sciences, Australia
andreas.ernst@csiro.au

Abstract. The travelling salesman problem with time windows is a difficult optimization problem that appears, for example, in logistics. Among the possible objective functions we chose the optimization of the makespan. For solving this problem we propose a so-called Beam-ACO algorithm, which is a hybrid method that combines 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. In this work we use stochastic sampling as an alternative to bounding information. Our results clearly demonstrate that the proposed algorithm is currently a state-of-the-art method for the tackled problem.

1 Introduction

The travelling salesman problem with time windows (TSPTW) [1] seeks to find an efficient route to visit a number of customers, starting and ending at a depot, with the added difficulty that each customer may only be visited within a certain time window. In practice, the TSPTW is an important problem in logistics. The TSPTW is proven to be NP-hard, and even finding a feasible solution is an NP-complete problem [2]. The problem is closely related to a number of important problems. For example, the well-known travelling salesman problem (TSP) is a special case of the TSPTW. The TSPTW itself can be seen as a special case with a single vehicle of the vehicle routing problem with time windows (VRPTW). The literature mentions two different objective functions for this problem. In this work we chose the optimization of the makespan as the objective. The ant

* 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.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 97–108, 2009.

colony optimization approaches from [3,4] are among the current state-of-the-art algorithms for the TSPTW when optimizing the makespan.

Ant colony optimization (ACO) is a metaheuristic that is based on the probabilistic construction of solutions [5]. At each algorithm iteration, a number of solutions are constructed independently of each other. A recently proposed ACO hybrid, known as Beam-ACO [6,7], employs at each iteration a probabilistic beam search procedure that constructs a number of solutions interdependently and in parallel. At each step, beam search keeps a certain number of the best partial solutions available for further extension [8]. 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. López-Ibáñez and Blum [9] presented a first study with the aim to show that *stochastic sampling* [10,11] is a useful alternative to bounding information. Hereby, each given partial solution is completed a certain number of times in a stochastic way. The information that is obtained in this way is used to differentiate between different partial solutions.

In this work, apart from comparing our results to the best known ones from the literature, we also study the effect that different components of Beam-ACO with stochastic sampling have on the performance of the algorithm. In particular, we will evaluate the influence of the pheromone information and the effects of different degrees of stochastic sampling. The remainder of this work is organized as follows. In Section 2 we give a technical description of the TSPTW. Section 3 introduces the Beam-ACO algorithm for 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

The TSPTW is formally defined as follows. 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 problem 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 sub-sequence $(p_1, \ldots, p_k, \ldots, p_n)$ is a permutation of the nodes in $N \setminus \{0\}$ and p_k denotes the index of the customer at the k^{th} position of the tour. Two additional elements, $p_0 = 0$ and $p_{n+1} = 0$, represent the starting depot and the final depot.

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 the customer p_k in the tour.

Two different but related objectives for this problem are found in the literature. One is the minimization of the cost of the edges traversed along the tour. The other alternative is to minimize $A_{p_{n+1}}$, that is, the arrival time at the depot. The first objective is analogous to the objective of the TSP, while the second is similar to the concept of a *makespan* in scheduling problems. In this paper, we focus on the latter. Hence, we formally define the TSPTW as:

$$\min F(P) = A_{p_{n+1}} \ , \tag{1}$$

subject to $\Omega(P) = \sum_{k=0}^{n+1} \omega(p_k) = 0$, where $\omega(p_k) = 1$ if $A_{p_k} > l_{p_k}$, and 0 otherwise. Note that $A_{p_{n+1}}$ is recursively computed as $A_{p_{k+1}} = \max(A_{p_k}, e_{p_k}) + c(a_{p_k,p_{k+1}})$. 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 first explain the solution construction, which is the crucial part of our Beam-ACO algorithm. The application of the (Beam-)ACO framework to any problem implies the definition of a pheromone model $\mathcal{T}$ and a solution construction mechanism. In fact, we need a solution construction mechanism for the probabilistic beam search as well as for stochastic sampling.

Stochastic Sampling. Given a partial solution, an ant a chooses at each construction step one customer j among the set $\mathcal{N}(P_a)$ of customers not included yet in the current partial tour P_a. Once all customers have been added to the tour, it is completed by adding node 0. The decision of which customer to choose at each step is done with the help of pheromone information and heuristic information. As for the pheromone information, $\forall a_{ij} \in A$, $\exists \tau_{ij} \in \mathcal{T}$, $0 \leq \tau_{ij} \leq 1$, where τ_{ij} represents the desirability of visiting customer j after customer i in the tour. The greater the pheromone value τ_{ij}, the greater is the desirability of choosing j as the next customer to visit in the current tour.

The decision of which customer to choose is made 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 is chosen deterministically as the value with the highest product of pheromone and heuristic information, that is, $j = \arg\max_{k \in \mathcal{N}(P_a)}\{\tau_{ik} \cdot \eta_{ik}\}$, where i is the last customer added to the tour P_a, and η_{ij} is the heuristic information that represents an estimation of the benefit of visiting customer j after customer i. Otherwise, j is stochastically chosen from the following distribution of probabilities:

$$\mathbf{p}_i(j) = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{k \in \mathcal{N}(P_a)} \tau_{ik} \cdot \eta_{ik}} \quad \text{if } j \in \mathcal{N}(P_a) \tag{2}$$

There are several heuristics that could be used for the TSPTW. When deciding which customer should be visited next, not only a small travel cost between customers (c_{ij}) 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 the travel cost between customers, the latest service time (l_j) and the earliest service time (e_j). The values are first normalized to $[0, 1]$, with the maximum value corresponding to 0 and the minimum to 1, and then combined:

$$\eta_{ij} = \lambda^c \frac{c^{\max} - c_{ij}}{c^{\max} - c^{\min}} + \lambda^l \frac{l^{\max} - l_j}{l^{\max} - l^{\min}} + \lambda^e \frac{e^{\max} - e_j}{e^{\max} - e^{\min}} \tag{3}$$

where $\lambda^c + \lambda^l + \lambda^e = 1$ are weights that allow to balance the importance of each heuristic. In earlier experiments, we found out that no single combination of weights would perform optimally across all instances in our benchmark set. Therefore, we decided to define the weights randomly for each application of probabilistic beam search.

The solution construction mechanism described above may result in the construction of infeasible solutions. Therefore, it is necessary to define a way of comparing between different—possibly infeasible—solutions. This will be done lexicographically ($<_{lex}$) by first minimising 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:

$$P <_{\text{lex}} P' \iff \Omega(P) < \Omega(P') \vee (\Omega(P) = \Omega(P') \wedge F(P) < F(P')) \tag{4}$$

Probabilistic Beam Search. The probabilistic beam search that we developed for the TSPTW is described in Algorithm 1. The algorithm requires three input parameters: $k_{\text{bw}} \in \mathbb{Z}^+$ is the so-called *beam width*, $\mu \in \mathbb{R}^+ \geq 1$ is a parameter that determines 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 tours called the *beam*. Hereby, index t denotes the current iteration of the beam search. At any time it holds that $|B_t| \leq k_{\text{bw}}$, that is, the beam is smaller than or equal to the beam width. A problem-dependent greedy function $\nu()$ is utilized to assign a weight to partial solutions.

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 henceforth denoted by $\langle P, j\rangle$. At each iteration, at most $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ candidate extensions are selected from C by means of the procedure $\mathsf{ChooseFrom}(C)$ to form the new beam B_{t+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 the current iteration is equal to the number of customers ($t = n$), all elements in B_n are completed by adding the depot, and finally the best solution is returned.

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

```
1: B_0 := {(0)}
2: randomly define the weights λ^c, λ^l, and λ^e
3: for t := 0 to n do
4:    C := C(B_t)
5:    for k = 1, ..., min{⌊μ · k_bw⌋, |C|} do
6:       ⟨P, j⟩ := ChooseFrom(C)
7:       C := C \ ⟨P, j⟩
8:       B_{t+1} := B_{t+1} ∪ ⟨P, j⟩
9:    end for
10:   B_{t+1} := Reduce(B_{t+1}, k_bw)
11: end for
12: output: arg min_lex {T | T ∈ B_n}
```

The procedure $\mathsf{ChooseFrom}(C)$ 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 $\mathsf{ChooseFrom}(C)$, 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 $\mathcal{T}$ and the greedy function $\nu()$: $\langle P, j\rangle = \arg\max_{\langle P',k\rangle \in C} \tau(\langle P', k\rangle) \cdot \nu(\langle P', k\rangle)^{-1}$, 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:

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

The greedy function $\nu(\langle P, j\rangle)$ assigns a heuristic value to each candidate extension $\langle P, j\rangle$. In principle, for this purpose we could use the heuristic η given by Eq. (3), that is, $\nu(\langle P, j\rangle) = \eta(\langle P, j\rangle)$. As in the case of the pheromone information, the notation $\eta(\langle P, j\rangle)$ refers to the value of η_{ik} as defined in Eq. (3), supposing that i was the last customer visited in tour 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 $\nu()$ 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 $\nu(\langle(1), 2\rangle) = 1 + 2 = 3$.

Finally, the application of procedure $\mathsf{Reduce}(B_t)$ removes the worst $\max\{|B_t| - k_{\text{bw}}, 0\}$ 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 the paragraph above

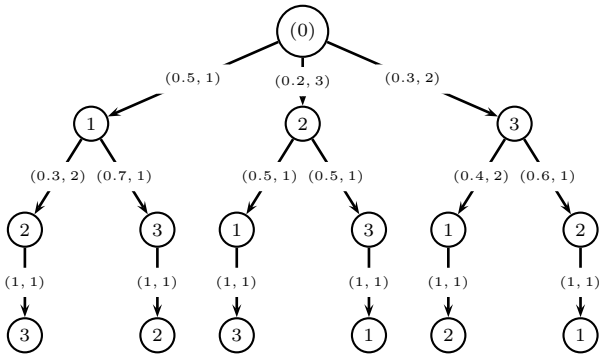

Fig. 1. Search tree corresponding to a problem instance with three customers. Edge labels are tuples that contain the heuristic information (η) in the first place, and the corresponding rank in the second place.

on stochastic sampling. The value of the best of these samples (with respect to Eq. 4) 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.

3.1 Beam-ACO Framework

The probabilistic beam search outlined in the previous section is used to construct solutions within an ACO algorithm that is implemented in the hyper-cube framework [12]. A high level description of the algorithm is given in Algorithm 2. The data structures used, in addition to counters and to the pheromone values, 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 *convergence factor* (*cf*), $0 \leq cf \leq 1$, which is a measure of how far the algorithm is from convergence; and (*4*) the Boolean variable *bs_update*, which becomes true when the algorithm reaches convergence.

Roughly, the algorithm works as follows. Initially, all variables are initialized. In particular, the pheromone values are set to their initial value 0.5. Then, a main loop is repeated until a termination criteria, such as a CPU time limit, is met. Each algorithm iteration consists of the following steps. First, a probabilistic beam search algorithm is executed, which returns solution P^{ib}. Then, 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 two remaining procedures of Algorithm 2 in more detail.

Algorithm 2. ACO algorithm for the TSPTW

input: $N^{\mathrm{s}}, k_{\mathrm{bw}} \in \mathbb{Z}^+$, $\mu \in \mathbb{R}^+$, $q_0 \in [0,1] \subset \mathbb{R}$
$P^{\mathrm{bf}} :=$ NULL, $P^{\mathrm{rb}} :=$ NULL, $cf := 0$, *bs_update* := FALSE
$\tau_{ij} := 0.5 \quad \forall \tau_{ij} \in \mathcal{T}$
while CPU time limit not reached **do**
  $P^{\mathrm{ib}} :=$ PBS(k_{bw},μ,N^{s}) /* see Algorithm 1 */
  if $P^{\mathrm{ib}} <_{\mathrm{lex}} P^{\mathrm{rb}}$ **then** $P^{\mathrm{rb}} := P^{\mathrm{ib}}$
  if $P^{\mathrm{ib}} <_{\mathrm{lex}} P^{\mathrm{bf}}$ **then** $P^{\mathrm{bf}} := P^{\mathrm{ib}}$
  $cf :=$ ComputeConvergenceFactor($\mathcal{T}$)
  if *bs_update* = TRUE **and** $cf > 0.99$ **then**
    $\tau_{ij} := 0.5 \quad \forall \tau_{ij} \in \mathcal{T}$
    $P^{\mathrm{rb}} :=$ NULL, *bs_update* := FALSE
  else
    if $cf > 0.99$ **then** *bs_update* := TRUE **end if**
    ApplyPheromoneUpdate(cf, *bs_update*, $\mathcal{T}$, P^{ib}, P^{rb}, P^{bf})
  end if
end while
output: P^{bf}

Procedure ComputeConvergenceFactor($\mathcal{T}$) 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 $(0, 1)$.

The next step of the algorithm updates the pheromone information by means of the procedure 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:

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

with $\xi_{ij} = \kappa^{\mathrm{ib}} \cdot P^{\mathrm{ib}}_{ij} + \kappa^{\mathrm{rb}} \cdot P^{\mathrm{rb}}_{ij} + \kappa^{\mathrm{bf}} \cdot P^{\mathrm{bf}}_{ij}$, 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 $\kappa^{\mathrm{ib}} + \kappa^{\mathrm{rb}} + \kappa^{\mathrm{bf}} = 1$. 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 $\tau^{\max}$ (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.

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

4 Experimental Evaluation

We implemented Beam-ACO in C++ and used 30 instances originally provided by Potvin and Bengio [13] for testing. These instances are known to contain a mix of randomly-located and clustered customers. First, we performed a set of initial experiments in order to find appropriate values for various parameters of Beam-ACO. On the basis of these experiments we chose $k_{\text{bw}} = 10$, $\mu = 1.5$, $N^{\text{s}} = 5$, $q_0 = 0.9$, and a time limit of 60 CPU seconds per run and per instance. Each experiment was repeated 25 times with different random seeds. All experiments were run on a AMD Opteron 8218 processor, with 2.6 GHz CPU and 1 MB of cache size running GNU/Linux 2.6.24.

Comparison to the State-of-the-art. In Table 2 we compare the results of Beam-ACO with the results of two ACO algorithms proposed in the literature: ACS-TSPTW [3] and ACS-Time [4], where ACS-TSPTW is a variation of ACS-Time. These algorithms can be regarded as the current state-of-the-art for the TSPTW with makespan optimization. The structure of Table 2 is as follows. $\tilde{F}$ is the mean makespan obtained by each algorithm, and T_{CPU} is the mean computation time in seconds. The results of ACS-TSPTW and ACS-Time were obtained using an AMD Athlon CPU with 1.46 GHz, which should be about two times slower than our machine. For the results of Beam-ACO, we provide the corresponding standard deviations ("sd"). Finally, the column "Old Best-known" gives the previously best known result for each instance, while "New Best-known" gives the best-known makespan taking into account the results obtained by Beam-ACO.

Beam-ACO obtained a feasible solution in all 25 runs, while it is not clear how many feasible solutions were obtained by ACS-TSPTW and ACS-Time. In case no feasible solution was obtained in any of the 5 runs of ACS-TSPTW or ACS-Time, the corresponding entry is blank. In fact, ACS-TSPTW fails to find any feasible solution in three cases, whereas ACS-Time fails to find any feasible solution in five cases. Beam-ACO achieves a better performance than the other two algorithms in 22 out of 30 cases. In further seven cases our algorithm equals the best of the results obtained by the other two algorithms. Only for one instance (rc.202.3) Beam-ACO was not able to obtain the best result known. In many

Table 2. Comparison of results obtained by ACS-TSPTW, ACS-Time and Beam-ACO

		ACS-TSPTW		ACS-Time		Beam-ACO				Best-known	
Problem	n	$\tilde{F}$	T_{CPU}	$\tilde{F}$	T_{CPU}	$\tilde{F}$	sd	T_{CPU}	sd	Old	New
rc201.1	20	592.06	100.94	592.06	96.77	592.06	0.00	0.01	0.00	592.06	592.06
rc201.2	26	877.49	246.26	866.56	262.66	860.17	0.00	0.39	0.44	861.91	860.17
rc201.3	32	867.61	464.30	854.11	466.13	853.71	0.00	0.09	0.15	853.71	853.71
rc201.4	26	900.52	151.05	—	—	889.18	0.00	0.16	0.32	900.38	889.18
rc202.1	33	880.74	241.77	880.74	241.72	850.48	0.00	0.95	1.15	871.11	850.48
rc202.2	14	338.52	46.77	382.47	47.17	338.52	0.00	0.00	0.00	338.52	338.52
rc202.3	29	892.18	190.24	—	—	894.10	0.00	0.26	0.50	847.31	847.31
rc202.4	28	—	—	874.55	170.75	853.71	0.00	1.18	1.19	856.37	853.71
rc203.1	19	673.07	78.83	600.66	80.44	488.42	0.00	0.01	0.01	572.63	488.42
rc203.2	33	926.75	255.77	911.34	278.96	853.71	0.00	4.05	3.37	897.88	853.71
rc203.3	37	—	—	—	—	921.54	0.51	20.47	13.58	—	921.44
rc203.4	15	493.85	53.08	429.96	52.11	338.52	0.00	0.01	0.01	415.26	338.52
rc204.1	46	949.68	438.25	—	—	925.12	1.14	24.79	15.62	949.22	920.11
rc204.2	33	863.65	240.55	770.08	238.42	691.58	3.21	22.46	15.74	753.52	690.06
rc204.3	24	642.06	127.27	533.25	128.80	456.19	1.58	19.15	18.71	488.36	455.03
rc205.1	14	422.24	46.90	421.57	46.67	417.81	0.00	0.03	0.05	417.81	417.81
rc205.2	27	820.19	181.06	820.19	195.11	820.19	0.00	3.44	2.68	820.19	820.19
rc205.3	35	950.05	274.55	951.22	273.09	950.05	0.00	0.13	0.20	950.05	950.05
rc205.4	28	870.43	186.56	849.32	180.18	837.71	0.00	1.70	1.09	838.75	837.71
rc206.1	4	117.85	13.27	117.85	13.41	117.85	0.00	0.00	0.00	117.85	117.85
rc206.2	37	914.99	306.13	906.98	304.27	879.17	6.07	14.22	16.85	905.47	870.49
rc206.3	25	650.59	140.72	650.59	140.51	650.59	0.00	0.01	0.01	650.59	650.59
rc206.4	38	943.31	320.19	—	—	920.18	5.17	29.44	18.14	943.31	911.98
rc207.1	34	860.98	258.44	889.33	258.28	820.23	4.19	26.88	17.07	851.06	809.86
rc207.2	31	—	—	792.38	—	720.78	1.07	24.21	16.76	761.78	717.22
rc207.3	33	955.70	241.70	844.98	233.68	757.80	8.49	34.73	13.39	836.05	747.47
rc207.4	6	133.14	22.45	133.14	22.80	133.14	0.00	0.00	0.00	133.14	133.14
rc208.1	38	934.80	334.72	901.61	331.58	820.88	8.84	41.96	10.42	877.20	810.70
rc208.2	29	722.24	185.73	608.84	185.33	581.32	0.00	8.50	5.89	591.43	581.32
rc208.3	36	795.03	291.98	739.54	295.94	691.66	1.30	26.67	15.14	715.27	686.80

instances, Beam-ACO found the (presumably) optimal solution in all 25 runs, as illustrated by a zero standard deviation.

With respect to computation time, Beam-ACO is between 5 and 100 times faster than the other two algorithms. This difference of computation time between Beam-ACO and the other algorithms cannot be explained solely by differences in processor speed.

Summarizing, the results let us conclude that Beam-ACO is a new state-of-the-art algorithm for the TSPTW with makespan optimization.

Analysis of Beam-ACO. With the aim of obtaining a better understanding of the behaviour of Beam-ACO we conducted a series of additional experiments. First, we wanted to study the influence and the importance of the pheromone information, which is used during the construction process of probabilistic beam search

as well as for stochastic sampling. For that purpose we repeated the 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 **noph**. In a second set of experiments we wanted to study the importance of stochastic sampling. Remember that, at each step of the probabilistic beam search, first a number of maximally $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ extensions 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 where $\mu = 1$ and $k_{\text{bw}} = 15$ (in order to compensate for the smaller μ value). Note that when $\mu = 1$, procedure Reduce is never invoked and stochastic sampling is never performed. In the presentation of the results this version of Beam-ACO is denoted by **no_ss**. Finally, we wanted to study how good stochastic sampling is in terms of an estimate, by applying it only after a certain number of iterations of each probabilistic beam search, that is, once the partial solutions in the beam of a probabilistic beam search contain a certain percentage of customers. More specifically, for the first $(n - (r^{\text{s}} \cdot n)/100)$ iterations of probabilistic beam search, stochastic sampling is not used. Instead, Reduce simply selects k_{bw} partial solutions at random. In contrast, for the remaining $(r^{\text{s}} \cdot n)/100$ iterations 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: $r^{\text{s}} = \{0\%, 25\%, 50\%, 75\%, 85\%, 100\%\}$. In the presentation of the results the corresponding algorithm versions are simply denoted by the value of parameter r^{s}.

Figure 2 shows the results of the different experiments described above for five problem instances that are rather difficult to solve. The barplots (in grey) compare the results with respect to the mean ranks obtained by each algorithm version over 25 runs. Note that standard deviations are shown as error bars. The ranks are calculated by sorting all solutions lexicographically. On the other hand, the boxplots (in white) show the distribution of computation time (in seconds) required by each algorithm version.

The following conclusions can be drawn. First, when no pheromone information is used (see algorithm **noph**), the performance of the algorithm drops significantly. Interestingly, the performance of Beam-ACO without pheromone update is always worse than the performance of Beam-ACO using at least $r^{\text{s}} = 50\%$ of stochastic sampling. Second, the use of stochastic sampling seems essential to achieve satisfactory results. When no stochastic sampling is used (see algorithm **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.

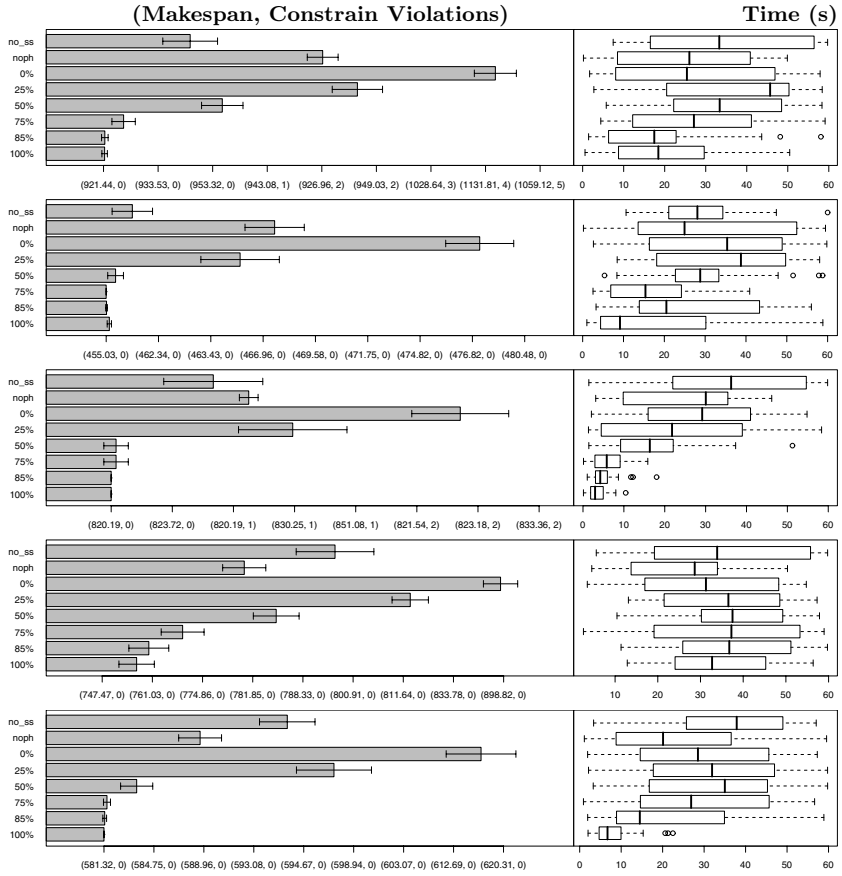

Fig. 2. Results concerning the analysis of Beam-ACO. From top to bottom the graphics concern instances rc.203.3, rc.204.3, rc.205.2, rc.207.3, and rc.208.2.

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. Disabling stochastic sampling completely ($r^{\mathrm{s}} = 0\%$), strongly affects the performance of Beam-ACO in a negative way. Starting from rates of stochastic sampling of at least 75%, the performance of the algorithm is already very close to—and sometimes the same as—the performance of Beam-ACO when always using stochastic sampling. However, even in those cases where a rate of stochastic sampling of 85% performs as well as the variant using $r^{\mathrm{s}} = 100\%$, the latter requires less computation time; see, for example, instances rc.204.3 and rc.208.2. This is particularly interesting because the variant using $r^{\mathrm{s}} = 100\%$ is the most expensive one in terms of computational effort. Therefore, this result suggests that stochastic sampling helps the algorithm to converge faster to the best solutions.

5 Conclusions

In this paper, we have proposed a Beam-ACO approach for the TSPTW with makespan optimization. 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. Experiments were performed on a set of standard benchmark instances for the TSPTW, comparing the Beam-ACO approach to the best known methods from the literature. The results showed that Beam-ACO is a state-of-the-art algorithm for the TSPTW with makespan optimization. In a second set of experiments we analysed the influence of pheromone information and stochastic sampling on the performance of Beam-ACO. The results showed that both algorithmic components are essential for achieving high quality results. In the future we plan to improve the performance of our Beam-ACO approach further, for example, by the inclusion of local search.

References

1. Ohlmann, J.W., Thomas, B.W.: A compressed-annealing heuristic for the traveling salesman problem with time windows. INFORMS J. Comput. 19(1), 80–90 (2007)
2. Savelsbergh, M.W.P.: Local search in routing problems with time windows. Annals of Operations Research 4(1), 285–305 (1985)
3. Cheng, C.B., Mao, C.P.: A modified ant colony system for solving the travelling salesman problem with time windows. Mathematical and Computer Modelling 46, 1225–1235 (2007)
4. Gambardella, L., Taillard, E.D., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 63–76. McGraw Hill, London (1999)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
6. Blum, C.: Beam-ACO–hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comp. & Op. Res. 32, 1565–1591 (2005)
7. Blum, C.: Beam-ACO for simple assembly line balancing. INFORMS J. Comput. 20(4), 618–627 (2008)
8. Ow, P.S., Morton, T.E.: Filtered beam search in scheduling. Int. J. Prod. Res. 26, 297–307 (1988)
9. López-Ibáñez, M., Blum, C.: Beam-ACO based on stochastic sampling: A case study on the TSP with time windows. In: Battiti, R., et al. (eds.) Proceedings of LION3. LNCS. Springer, Berlin (2009)
10. Juillé, H., Pollack, J.B.: A sampling-based heuristic for tree search applied to grammar induction. In: Proceedings of AAAI 1998, pp. 776–783. MIT press, Cambridge (1998)
11. Ruml, W.: Incomplete tree search using adaptive probing. In: Proceedings of IJCAI 2001, pp. 235–241. IEEE press, Los Alamitos (2001)
12. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE T. Syst. Man Cyb. – Part B 34(2), 1161–1172 (2004)
13. Potvin, J.Y., Bengio, S.: The vehicle routing problem with time windows part II: Genetic search. INFORMS J. Comput. 8, 165–172 (1996)