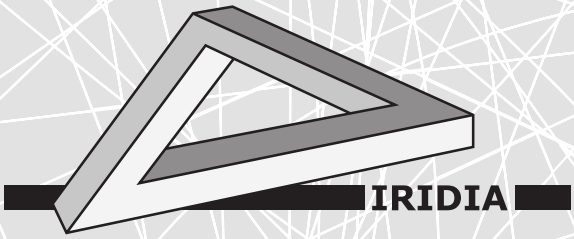

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

The Travelling Salesman Problem with Time Windows: Adapting Algorithms from Travel-time to Makespan Optimization

Manuel López-Ibáñez, Christian Blum, Jeffrey W. Ohlmann, and Barrett W. Thomas

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2013-011

June 2013

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2013-011

Published as: M. López-Ibáñez et al., The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization, Appl. Soft Comput. J. (2013), http://dx.doi.org/10.1016/j.asoc.2013.05.009

The Travelling Salesman Problem with Time Windows: Adapting Algorithms from Travel-time to Makespan Optimization☆

Manuel López-Ibáñez[a,*], Christian Blum[b], Jeffrey W. Ohlmann[c], Barrett W. Thomas[d]

[a] *IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium*
[b] *IKERBASQUE, Basque Foundation for Science, Bilbao, Spain*
and
Department of Computer Science and Artificial Intelligence, University of the Basque Country, San Sebastian, Spain
[c] *Department of Management Sciences, University of Iowa, Iowa City, USA*
[d] *Department of Management Sciences, University of Iowa, Iowa City, USA*

Abstract

In combinatorial optimization it is not rare to find problems whose mathematical structure is nearly the same, differing only in some aspect related to the motivating application. For example, many problems in machine scheduling and vehicle routing have equivalent formulations and only differ with respect to the optimization objective, or particular constraints. Moreover, while some problems receive a lot of attention from the research community, their close relatives receive hardly any attention at all. Given two closely related problems, it is intuitive that it may be effective to adapt state-of-the-art algorithms—initially introduced for the well-studied problem variant—to the less-studied problem variant. In this paper we provide an example based on the travelling salesman problem with time windows that supports this intuition. In this context, the well-studied problem variant minimizes the travel time, while the less-studied problem variant minimizes the makespan. Indeed, the results show that the algorithms that we adapt from travel-time minimization to makespan minimization significantly outperform the existing state-of-the-art approaches for makespan minimization.

Keywords:
ant colony optimization, compressed simulated annealing, travelling salesman problem with time windows, hybridization

1. Introduction

As in the classical travelling salesman problem (TSP), the travelling salesman problem with time windows (TSPTW) requires the specification of a Hamiltonian cycle through a graph of nodes, but adds the requirement that each node must be visited within a predefined time window. For a routing application, the TSPTW represents the problem of finding an efficient route, starting and ending at a specified depot, that visits a set of customers, each one in his predefined time window. In the machine scheduling environment, the TSPTW can be used to model the problem of sequencing 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 routing context, the TSPTW objective is typically to minimize the sum of the travel times. In the machine scheduling context, the TSPTW objective is to minimize the makespan. Both versions of the TSPTW are proven to be NP-hard, and even finding feasible solutions is an NP-complete problem [1]. For convenience we henceforth refer to the TSPTW with travel-time minimization by TSPTW-TT, and to the TSPTW with makespan minimization by TSPTW-M.

1.1. Contribution of this Paper

Among the two above-mentioned problem versions, the TSPTW-TT has been more closely studied in the literature. In fact, very few algorithms have been proposed for the TSPTW-M. In this paper, we adapt two state-of-the-art algorithms for the TSPTW-TT, compressed annealing (CA) [2] and the Beam-ACO algorithm [3], to the TSPTW-M. These algorithms, by virtue of being metaheuristics, can be easily adapted to different variants of the same problem. Moreover, in this case, the structure of the problem remains the same despite the change in the objective function. We do not attempt to adapt other algorithms developed for the TSPTW-TT to the TSPTW-M. First, some of these algorithms heavily rely on heuristics specific to the TSPTW-TT. Second, since Beam-ACO and CA are the state-of-the-art for the TSPTW-TT, there is no a priori indication that other algorithms could become better than them when adapted to the TSPTW-M.

Concerning the experimental results, we present an exhaustive comparison between both adapted algorithms on a compre-

☆This work was supported by grant TIN2012-37930-C02-02 of the Spanish government. Manuel López-Ibáñez acknowledges support from the Belgian F.R.S.-FNRS, of which he is a postdoctoral researcher.

*Corresponding author
Email addresses: manuel.lopez-ibanez@ulb.ac.be (Manuel López-Ibáñez), christian.c.blum@gmail.com (Christian Blum), jeffrey-ohlmann@uiowa.edu (Jeffrey W. Ohlmann), barrett-thomas@uiowa.edu (Barrett W. Thomas)

hensive set of benchmark instances available in the literature. This, as such, is a useful contribution because existing algorithm proposals for the TSPTW-M have only been applied to rather small subsets of the available benchmark instances. Furthermore, both algorithms are compared to current state-of-the-art algorithms developed specifically for the TSPTW-M.

Concerning the comparison between Beam-ACO and CA, we were able to detect—for all considered sets of benchmark instances—a statistically significant advantage of Beam-ACO over CA. With respect to the comparison of Beam-ACO and CA with existing state-of-the-art algorithms, the following conclusions can be drawn. First, for the benchmark set proposed by Dumas et al. [7], Beam-ACO is able to improve the best-known results from [12] in five out of 22 cases. In the remaining cases, the same results as in [12] are achieved. Currently, the state-of-the-art algorithms for the TSPTW-M are ACS-TSPTW and ACS-Time [16]. Both algorithms are based on the metaheuristic ant colony optimization. In [16], both algorithms were applied to the benchmark set proposed by Potvin & Bengio [22]. A comparison of Beam-ACO and CA with the two ACO algorithms on this benchmark set reveals that Beam-ACO outperforms ACS-TSPTW in 17 out of 24 cases and ACS-Time in 20 out of 24 cases. While in the remainder cases both algorithms obtain the best-known solution in every run, Beam-ACO is from 40 to more than one hundred times faster than the other algorithms. The results of CA are similar to those of Beam-ACO.

1.2. Organization of the Paper

The remainder of the paper is organized as follows. In Section 2, we provide a technical description of the TSPTW. We review the history of both problem variants in Section 2.1. In Section 3, we briefly describe CA and Beam-ACO as well as providing a description of the changes that were necessary for adapting them to makespan optimization. The benchmark instances considered in this work are described in Section 4. After outlining the tuning process, we provide comprehensive results in Section 5 for both algorithms on a comprehensive set of benchmark instances used in the related literature. Finally, we offer conclusions and an outlook to future work in Section 6.

2. Technical Problem Description

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 TSPTW solution is a sequence visiting each node of G exactly once (starting and ending at the depot). We represent a tour 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 $N \setminus \{0\}$ and p_k denotes the index of the node at the k^{th} position of the tour. The two additional elements—that is, $p_0 = 0$ and $p_{n+1} = 0$—represent the depot at which the tour starts and ends.

For each edge $a_{ij} \in A$, which connects the two nodes i and j, there is an associated cost $c(a_{ij})$. In a routing application, $c(a_{ij})$ represents the service time at customer i plus the travel time between customer i and j. In a scheduling application, $c(a_{ij})$ represents the task time for job i plus the time to set up for job j when following job i. A time window $[e_i, l_i]$ is associated to each node $i \in N$, which defines when service at node i can begin. Waiting times are generally permitted; in the routing, this implies that a customer i may be reached before the start e_i of its time window, but service cannot start until e_i. Therefore, given a particular tour P, the departure time from node p_k is calculated as

$$D_{p_k} = \max(A_{p_k}, e_{p_k}) \ , \tag{1}$$

where $A_{p_k} = D_{p_{k-1}} + c(a_{p_{k-1},p_k})$ is the arrival time at node p_k in the tour. A tour P is feasible, if and only if $\Omega(P) = \sum_{k=0}^{n+1} \omega(p_k) = 0$, where $\omega(p_k) = 1$ if $A_{p_k} > l_{p_k}$, and 0 otherwise.

As outlined above, two different objective functions may be considered for the TSPTW. The function for travel-time minimization, $f_{\text{tt}}(\cdot)$, concerns the minimization of the sum of the costs of the edges traversed along a tour. That is, given a tour P:

$$f_{\text{tt}}(P) = \sum_{k=0}^{n} c(a_{p_k,p_{k+1}}) \tag{2}$$

Note that the objective function $f_{\text{tt}}(\cdot)$ is analogous to the one of the classical TSP. The other alternative is to minimize $A_{p_{n+1}}$, that is, the arrival time at the depot. That is, given a tour P:

$$f_{\text{m}}(P) = A_{p_{n+1}} \tag{3}$$

where function $f_{\text{m}}(\cdot)$ refers to makespan minimization.

2.1. Previous Work on the TSPTW

The earliest papers on the TSPTW focused on exact approaches for makespan optimization [4, 5]. The testing of these approaches was limited to instances of at most 50 nodes. Moreover, these algorithms were not able to handle wide or overlapping time windows. Langevin et al. [6] proposed a branch-and-bound scheme to solve a two-commodity flow formulation for the TSPTW and considered both makespan and travel-time optimization; their implementation was able to solve instances of up to 40 nodes. Dumas et al. [7] extended earlier dynamic programming approaches by using state space reduction techniques that allowed them to solve larger problem instances. More recently, Ascheuer et al. [8] developed a branch-and-cut algorithm in which they applied techniques tailored for the asymmetric TSPTW. Balas & Simonetti [9] presented a linear-time dynamic programming algorithm for several TSP variants with precedence constraints, including the TSPTW. Constraint programming is another exact approach that has been applied to the TSPTW [10, 11].

Due to the inherent difficulty of the TSPTW, heuristics have been the focus of research in more recent years. Carlton & Barnes [12] developed a tabu search approach to optimize a hierarchical objective that attempts to primarily minimize makespan, and then secondly to minimize total travel time subject to maintaining the minimal makespan. In order to deal with infeasibility, Carlton & Barnes [12] augment the objective function with a static penalty for violating l_i of customer i. Gendreau et al. [13] presented a constructive heuristic with a

subsequent improvement procedure. Wolfler Calvo [14] presented a constructive 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 obtained solution. Two state-of-the-art TSPTW-TT algorithms for travel-time optimization were proposed by Ohlmann & Thomas [2] and López-Ibáñez & Blum [3]. Ohlmann & Thomas [2] proposed a compressed annealing (CA) algorithm, which is a variant of simulated annealing making use of a variable penalty method. Finally, López-Ibáñez & Blum [3] presented Beam-ACO algorithm, which results from the combination of ant colony optimization (ACO) with beam search, a heuristic variant of branch-and-bound. The same authors [3] also provided a comprehensive comparison of CA and Beam-ACO. Recently, a variable neighborhood search approach has been proposed [15]. This algorithm appears to perform similarly well. However, it has not been tested on all available benchmark instances. Authors developing heuristic approaches have focused on travel-time optimization and fewer results are published on makespan optimization. Curiously, TSPTW-M has been the focus of the ant colony optimization community [16, 17, 18]. Some of these ACO approaches [16, 17] can be considered state-of-the-art for the TSPTW-M.

3. Adaptation of the Algorithms to Makespan Optimization

In the following we provide a short description of both CA (as published in [2]) and Beam-ACO (as published in [3]) for the TSPTW-M.

3.1. Compressed Annealing for the TSPTW-M

Compressed annealing (CA) incorporates a variable penalty approach within the stochastic local search framework of simulated annealing to address constrained combinatorial optimization problems (such as the TSPTW). Simulated annealing algorithms escape local optima by probabilistically allowing moves to non-improving neighbors. Simulated annealing controls the likelihood of accepting a non-improving neighbor solution via a parameter called temperature, τ. CA introduces an additional parameter called pressure, λ, which controls the likelihood of accepting an infeasible solution (with respect to the time window constraints) by introducing a penalty proportional to the degree of infeasibility. Specifically, CA evaluates each tour P according to the augmented function $v(P, \lambda) = f_m(P) + \lambda\rho(P)$, where

$$\rho(P) = \sum_{i=1}^{n} \left[\max\left\{0, D_{p_i} - l_{p_i}\right\}\right] \tag{4}$$

is the degree of time window violation. The search behavior of CA is governed by the manner in which temperature and pressure are adjusted during the annealing run. Every ι iterations, the values of temperature and pressure are respectively updated according to $\tau_{k+1} = \beta\tau_k$ and $\lambda_{k+1} = \hat{\lambda}\left(1 - e^{-\gamma k}\right)$, where β and γ are user-specified parameters.

To estimate the initial temperature value, τ_0, as well as $\hat{\lambda}$, we execute a pre-processing step that samples 5000 pairs of neighbor solutions (for a 1-shift neighborhood). Based on this sample, we compute

$$\tau_0 = \frac{\overline{|\Delta v|}}{\ln\left(\frac{1}{\chi_0}\right)}, \tag{5}$$

and

$$\hat{\lambda} = \max_P \left\{\frac{f_{\mathrm{m}}(P)}{\rho(P)}\right\} \frac{\kappa}{1-\kappa}, \tag{6}$$

where $\overline{|\Delta v|}$ is the average absolute difference in objective function over the 5000 sample transitions, and the values of χ_0 and κ are user-specified. CA is terminated when the best solution has not been updated in the last I temperature/pressure changes ($I \times \iota$ neighbor solutions) while requiring a minimum of 100 total temperature changes, i.e., the algorithm will consider at least $100 \times \iota$ neighbor solutions. We discuss the calibration of parameters ι, β, γ, χ_0, and κ in Section 5.1. See Algorithm 1 for an overview of CA; we refer to Ohlmann & Thomas [2] for further details on CA.

Algorithm 1 Pseudo-code of Compressed Annealing

```
Initialize best tour found, P_best, so that f_m(P_best) = ∞ and ρ(P_best) = 0
Generate initial tour, P
k := 0
Set initial temperature and pressure, τ_k and λ_k
Set ι, the number of iterations at each temperature/pressure
repeat
    counter := 0
    repeat
        counter := counter + 1
        Randomly generate P', a neighbor tour of P
        With probability exp(−(v(P',λ_k)−v(P,λ_k))^+ / τ_k), let P := P'
        if ρ(P) ≤ ρ(P_best) and f_m(P) < f_m(P_best) then
            P_best := P
        end if
    until counter = ι
    k := k + 1
    Update τ_k and λ_k according to cooling and compression schedules
until termination criterion satisfied
output: P_best
```

3.2. Beam-ACO for the TSPTW-M

Beam-ACO is an algorithm that results from the hybridization of the metaheuristic ant colony optimization (ACO) with beam search. The general framework of a Beam-ACO algorithm [19] is based on ACO. In ACO, a certain number of solutions are constructed at each iteration *independent of each other*. This is based on a so-called pheromone model, a set $\mathcal{T}$ of numerical values that are used to generate the probabilities for the possible extensions of the partial solution considered at the current construction step. In the case of the TSPTW, partial solutions correspond to partial tours. Moreover, an extension

of a partial tour is obtained by visiting exactly one of the so-far unvisited customers.

The general framework of Beam-ACO is shown in Algorithm 2. In contrast to ACO, Beam-ACO employs—at each iteration—a probabilistic beam search procedure that generates a number of k_{bw} solutions interdependently and in parallel (see line 5 of Algorithm 2). The best one of these solutions, P^{ib}, is provided as feedback to the algorithm. The pseudo-code of the probabilistic beam search that we devised is given in Algorithm 3. Parameter k_{bw} is known as the *beam width*. Just like in ACO, the extension of partial solutions in Beam-ACO is based on a greedy function and on the pheromone model. More specifically, based on greedy and pheromone information, Beam-ACO selects at each construction step a number of $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ extensions of the current set of k_{bw} partial solutions. Hereby, $\mu > 1$ is a parameter of the algorithm. As the number of solutions to be generated is limited by k_{bw}, the next step consists in reducing the set of selected partial solutions to the k_{bw} best ones. This is generally done by means of bounding information. Hence, accurate and inexpensive bounding information is of crucial importance for Beam-ACO. In other words, when the bounding information is either misleading or when this information is computationally expensive, Beam-ACO may not be the algorithm of choice. Unfortunately, this has turned out to be the case for the TSPTW. Therefore, López-Ibáñez & Blum [3] replaced, in the context of the TSPTW-TT, the use of bounding information by a technique known as *stochastic sampling* [20, 21].

Algorithm 2 Pseudo-code of Beam-ACO

1: Initialize best tour found, P_{best}, so that $f_{\text{m}}(P_{\text{best}}) = \infty$
2: Initialize parameters k_{bw}, μ and N^{s}
3: Initialize the set $\mathcal{T}$ of pheromone values
4: **repeat**
5: Generate a solution P^{ib} by *probabilistic beam search* based on parameters k_{bw}, μ, N^{s} and $\mathcal{T}$
6: Apply local search to P^{ib}
7: **if** $P^{\text{ib}} <_{\text{lex}} P_{\text{best}}$ **then** $P_{\text{best}} := P^{\text{ib}}$
8: Apply an update of the pheromone values
9: **until** termination criterion satisfied
10: **output:** P_{best}

By means of stochastic sampling, partial solutions are evaluated in the following alternative way. Starting from the partial solution under consideration, a complete solution is probabilistically generated N^s times, based on greedy and pheromone information. Hereby, N^s is called the *number of samples*. The objective function value of the best of these N^s samples is then used to evaluate the corresponding partial solution. The information obtained by stochastic sampling is used to rank the different partial solutions. The worst partial solutions are then excluded from further examination. The importance (and accuracy) of stochastic sampling is controlled by an extra parameter, the *rate of stochastic sampling*, referred to by $r^{\text{s}} \in \{0, \ldots, 100\}$ (in percent). More specifically, for the first $(n - (r^{\text{s}} \cdot n)/100)$ construction steps of probabilistic beam search, stochastic sampling is not used at all. Instead, the k_{bw} partial solutions for further examination are chosen randomly from the set of $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ generated extensions. Stochastic sampling is only used in the remaining construction steps. Note that when $r^{\text{s}} = 0$ stochastic sampling is not applied at all, while $r^{\text{s}} = 100$ refers to the use of stochastic sampling at each construction step.

Algorithm 3 Pseudo-code of probabilistic beam search

1: Initilize set B_0 with a partial solution that only contains the depot
2: **for** $t = 0$ **to** n **do**
3: Generate the set of all extensions, C, of the partial solutions in B_t
4: $B_{t+1} := \emptyset$
5: **for** $k = 1$ **to** $\min\{\lfloor \mu \cdot k_{\text{bw}} \rfloor, |C|\}$ **do**
6: Choose one of the children from C, c, based on greedy/pheromone information
7: Remove c from C
8: Add c to B_{t+1}
9: **end for**
10: Reduce the size of B_{t+1} to k_{bw} by means of *stochastic sampling*
11: **end for**
12: **output:** The best solution from B_n

Finally, we adapt the 1-opt best-improvement local search already proposed for the TSPTW-TT [3]. This local search is applied to the best solution generated at each iteration. In the 1-opt neighborhood, a single customer is removed from the tour and reinserted in a different position. A similar local search was previously proposed by Carlton & Barnes [12], but with some notable differences. In particular, we implemented several speed-ups that are not found in previous proposals, and we compare solutions lexicographically, instead of using a penalty term. The adaptation of the 1-opt local search to the TSPTW-M requires that Beam-ACO keeps track of the partial makespan during solution construction. Since this was already required by the speed-ups proposed in the context of the TSPTW-TT [3], the local search algorithm remains basically the same.

Since we noticed that the best-improvement local search could be computationally expensive, we also implement here a first-improvement variant, which does not necessarily examine the whole neighborhood of a given solution. In the first-improvement variant, whenever a neighbor is found that is better than the current solution, this neighbor is selected without examining the rest of the neighborhood. Just like the best-improvement variant, the first-improvement variant stops when no improving neighbor can be found.

The Beam-ACO approach differs from other published algorithms for the TSPTW by not making use of a penalty term in the objective function. Instead it uses a lexicographic comparison of—possibly infeasible—solutions. For this purpose, an operator ($<_{\text{lex}}$) was defined that compares solutions by their number of constraint violations (Ω). In the case of an equal number of constraint violations, the objective function (f) is used as a second criterion. More formally, two different solu-

tions P and P' are compared as follows:

$$P <_{\text{lex}} P' \iff \Omega(P) < \Omega(P') \quad \textbf{or } (\Omega(P) = \Omega(P') \textbf{ and } f(P) < f(P')) \quad (7)$$

The interested reader may note that the pseudo-codes that are given above are slightly simplified. A complete description of the Beam-ACO algorithm can be found in the original paper [3].

4. Benchmark Instances

In the following, we briefly describe the seven benchmark sets considered in this work.[1]

1. The first set consists of 30 instances originally provided by Potvin & Bengio [22] and derived from Solomon's RC2 VRPTW instances [23]. 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. [6], 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. [7]. Instance size ranges from 20 to 200 customers.
4. Gendreau et al. [13] provided the fourth benchmark set consisting of 120 instances grouped into 26 classes with the same number of customers and equivalent time window widths. These instances were obtained from the instances proposed by Dumas et al. [7] 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 & Thomas [2], contains 25 instances grouped into five classes. The instances were derived from the instances with 150, respectively 200, customers proposed by Dumas et al. [7] by extending the time windows by 100 time units.
6. The sixth benchmark set consists of 50 asymmetric TSPTW instances introduced by Ascheuer [24]. These are real-world instances based on the routing of a stacker crane in an automated storage system.
7. Finally, the seventh benchmark set contains 27 symmetric instances proposed by Pesant et al. [10] (and also used by Focacci et al. [11]). While they were also derived from Solomon's RC2 VRPTW instances [23], they are different from the instances proposed by Potvin & Bengio [22].

[1] These instances are available at `http://iridia.ulb.ac.be/~manuel/tsptw-instances`

Table 1: Parameters of Beam-ACO and their values considered for tuning.

Parameter	Role	Domain
k_{bw}	beam width	$\{1, 2, \ldots, 9, 10\}$
μ	used for calculating the number of extensions to be chosen	$\{1, 1.1, \ldots, 4.9, 5\}$
r^s	sampling rate	$\{0, 1, \ldots, 99, 100\}$
N^s	number of samples	$\{1, 2, 3, 4, 5\}$
local search mode	first-improvement or best-improvement local search	{first, best}

5. Experimental Evaluation

We implemented Beam-ACO and CA in `C++`, compiled with `gcc` version 4.4. All experiments were performed on Intel Xeon E5410 CPUs, running at 2.33 Ghz with 6MB of cache under Cluster Rocks Linux version 4.2.1/CentOS 4. The implementation is sequential and experiments run on a single core.

5.1. Parameter Tuning

All benchmark sets described in the previous section are randomly partitioned in two distinct sets: 20% of the instances are used for tuning and the remaining 80% are used for testing. The parameters of both Beam-ACO and CA are tuned by means of Iterated F-race [25], as implemented by the `irace` software package [26]. Tables 1 and 2 indicate the domain of the parameter settings considered for tuning.

Each run of the tuner (`irace`) produces a single parameter configuration of the algorithm being tuned (either Beam-ACO or CA). The stopping criterion for a single run of the tuner is 1000 runs of the algorithm being tuned, and each individual run of Beam-ACO or CA is stopped after 60 seconds. The tuner does not directly use the objective function to evaluate the quality of each run of Beam-ACO or CA. Instead, in order to take into account the number of constraint violations (Ω), the makespan (f_{m}) and the computation time (T_{CPU}), the tuner uses the following formula to evaluate the cost value (Cost) of a run:

$$\text{Cost} = c_1 \cdot \Omega + c_2 \cdot f_{\text{m}} + T_{\text{CPU}} \quad (8)$$

where c_1 and c_2 are constants that were chosen as follows. First, given a run having produced a solution with a lower number of constraint violations than another run, the former should always have a lower cost value than the latter. This can be achieved by choosing c_1 to be (i) greater than any possible makespan value of any possible solution to any of the considered problem instances, and (ii) greater than any possible computation time. It was verified that $c_1 = 10^7$ satisfies this requirement concerning the set of benchmark instances that was tackled in this work. Second, in the case of two runs having produced solutions with the same number of constraint violations, the run with the better solution concerning the makespan value should have a lower cost value than the other run. This can be achieved by choosing c_2 to be greater than any possible computation time. As the computation time limit for all runs was 60 seconds, the setting $c_2 = 100$ satisfies this requirement.

Table 2: Tuning domain of CA parameters.

Parameter	Domain
β	$\{0.9, 0.91, \ldots, 0.99\}$
χ_0	$\{0.89, 0.9, \ldots, 0.99\}$
γ	$\{0.01, 0.02, \ldots, 0.11\}$
κ	$\{0.90, 0.91, \ldots, 0.9999\}$
I	$\{50, 51, \ldots, 150\}$
ι	$\{10000, 10001, \ldots, 50000\}$

We repeat each run of the tuner five times for each algorithm, thus, we obtain five parameter configurations of Beam-ACO and five parameter configurations of CA. Table 3 describes the configurations of Beam-ACO found by the five tuning runs, and Table 4 describes the ones corresponding to the tuning of CA. The table also includes the default parameter settings of CA (row "default") for further comparison. The parameter settings found by the different runs of the tuner are quite similar. In the case of Beam-ACO, they are characterized by small beamwidth, low μ, and the use of first-improvement local search. The values of sample rate and number of samples are more varied, which indicates some trade-off between these two parameters. The variability of the parameter settings found by the five tuning runs is also small in the case of CA. However, there are large differences with respect to the default parameter settings, in particular for ι and I. The next step is to test these configurations on the set of testing instances.

Table 3: Results of the automatic configuration of Beam-ACO.

Configuration number	Parameter setting				
	k_{bw}	μ	s^r	N^s	local search mode
1	1	4.13	91	4	first
2	3	1.37	19	2	first
3	2	2.15	12	3	first
4	2	1.21	56	4	first
5	2	1.22	65	3	first

Table 4: Results of the automatic configuration of CA.

Configuration number	Parameter setting					
	β	χ_0	γ	κ	I	ι
default	0.96	0.94	0.06	0.9999	75	30000
1	0.9	0.93	0.08	0.99	75	10087
2	0.90	0.96	0.1	0.99	112	10052
3	0.90	0.91	0.09	0.98	147	10519
4	0.9	0.9	0.08	0.99	123	10805
5	0.9	0.90	0.10	0.99	133	10801

5.2. Testing of the Tuned Configurations

We ran the above eleven algorithm versions (five settings of Beam-ACO and five settings of CA plus its default setting) on the remaining 80 percent of the benchmark instances that were not used during the tuning procedure. We repeat each run 15 times with a different random seed and stop each run after 60 seconds. We record the number of constraint violations (Ω), the relative percent deviation (RPD) of the makespan, that is, $100 \cdot (\text{makespan} - \text{best-known})/\text{best-known}$, and the CPU time ($T_{\text{CPU}}$) required to find the best solution returned by each run. For the sake of brevity, we provide here only a statistical summary of the results.

For each run, we calculate a single value using Eq. 8, and we average over the 15 repetitions. We apply the nonparametric Friedman test to the average of these values over the 15 repetitions of each experiment. The different configurations are the treatment factor and the instances within each benchmark set are the blocking factor. The Friedman test ranks the configurations for each instance: the lower the rank the better. Moreover, it calculates the sum of the ranks for each configuration. Then, given a confidence level—95 percent, for instance—it provides the minimum difference of ranks that is significant at that level ($\Delta R_{95\%}$). Tables 5 and 6 summarise the results of the Friedman test for each benchmark set. The different configurations are sorted according to their difference of ranks (ΔR) with respect to the best configuration. The value of ΔR is given in parenthesis and it should be compared with the critical value ($\Delta R_{95\%}$). When the critical value is infinite, there is no significant difference between the configurations. For ease of reading, we mark with a grey background those configurations that are not significantly different from the best one. In the case of Beam-ACO, Table 5 shows that configurations 1, 4, and 5 are most often the best ranked. Nonetheless, in most cases, there are no statistically significant differences between most configurations. In the case of CA, configurations 1, 4 and 5 are the best ranked,[2] whereas the default configuration is often among the worst ranked. Nonetheless, in almost all cases, the observed differences are not significant according to the Friedman test. These results indicate that the multiple runs of the tuning procedure found parameter settings of roughly the same quality.

5.3. Results of Beam-ACO and CA

For the sake of brevity, in the following we compare one configuration of Beam-ACO and one of CA. Specifically, we choose the best ranked configuration for each benchmark set, as reported in Tables 5 and 6. We report for each instance, the best-known makespan value (*Best*), the percentage of runs of an algorithm that did not find any feasible solution (*%inf*), the mean and standard deviation of the RPD makespan (*RPDm* and *RPDsd*) over 15 runs with different random seed, and the mean and standard deviation of the CPU time in seconds (*Tm* and *Tsd*) required to find the best solution returned by each run.

Table 7 concerns the results obtained for the asymmetric instances proposed by Ascheuer [24]. Except for two large instances, Beam-ACO is able to achieve the best-known solution in all runs in very short time. For these two large instances (`rbg193.tw` and `rbg233.tw`), Beam-ACO finds the

[2] It is by chance that the numbers of the best-ranked configurations of CA match those of Beam-ACO, since individual runs of the tuner are completely independent.

Table 5: Friedman test of Beam-ACO configurations per instance set (see text for details).

Benchmark	ΔR_α	**Configuration** (ΔR)
Ascheuer	14.64	#1 (0), #4 (12.5), #5 (14), #2 (16.5), #3 (27)
Dumas et al.	10.95	#4 (0), #5 (1), #2 (14.5), #1 (22), #3 (27.5)
Gendreau et al.	∞	#5 (0), #4 (0.5), #1 (3.5), #2 (4.5), #3 (11.5)
Ohlmann & Thomas	14.01	#5 (0), #4 (9.5), #2 (29.5), #1 (43), #3 (45.5)
Pesant et al.	∞	#4 (0), #1 (0.5), #5 (1), #3 (3.5), #2 (5)
Potvin & Bengio	∞	#1 (0), #5 (1), #4 (3.5), #2 (5), #3 (10.5)

Table 6: Friedman test of CA configurations per instance set (see text for details).

Benchmark	ΔR_α	**Configuration** (ΔR)
Ascheuer	∞	#5 (0), #4 (8.5), #1 (12.5), #2 (13.5), default (13.5), #3 (15)
Dumas et al.	∞	#4 (0), #5 (1), #2 (7), #3 (10.5), #1 (12), default (20.5)
Gendreau et al.	∞	#4 (0), #2 (8.5), default (12), #1 (14.5), #3 (19.5), #5 (20.5)
Ohlmann & Thomas	16.86	#1 (0), #3 (0), #5 (3.5), #2 (4), #4 (4), default (42.5)
Pesant et al.	∞	#4 (0), #5 (1), #3 (3.5), #1 (4), default (6), #2 (9.5)
Potvin & Bengio	∞	#5 (0), #2 (1.5), #3 (7.5), #1 (8), default (12.5), #4 (15.5)

best-known solution in 93.33% and 80% of the runs, respectively, while it fails to find a feasible solution in the rest. CA, on the other hand, is much faster on the largest instances, but it converges prematurely to slightly sub-optimal solutions for several instances of size 21. This suggests that structural differences between the instances play a larger role than instance size for what concerns the performance of Beam-ACO and CA. Apart from these few instances, the performance of Beam-ACO and CA is similar.

Table 8 shows results for the instances proposed by Dumas et al. [7]. In order to enable a comparison with previous results reported by Carlton & Barnes [12], the statistics of 15 applications to each instance are averaged over the five instances of each instance class. In this case, Beam-ACO generates the best-known solution in all runs for all but two instance classes, whereas CA fails to obtain a feasible solution in five instance classes. Although CA converges faster than Beam-ACO for instances of size $n \geq 150$, it often converges to an infeasible solution. In comparison to the results provided by Carlton & Barnes [12], we found new best-known solutions for five instance classes.

Table 9 compares the results of Beam-ACO with CA for the instances proposed by Gendreau et al. [13]. Following Gendreau et al. [13], we compute the statistics of 15 applications to each instance averaged over the five instances of each instance class. Hence, each instance class contains both tuning and testing instances. These instances have wide time windows for which the performance of exact algorithms tends to degrade. Interestingly, Beam-ACO is always able to find the best-known solution within one second. CA also finds very good feasible tours in short time. However, the quality of the solutions found by CA is slightly inferior to the ones obtained by Beam-ACO on a number of instances.

Table 10 examines the performance of Beam-ACO on the instances proposed by Ohlmann & Thomas [2]. In this case, we only use the test instances for comparison. These instances were designed to be difficult for both heuristic and exact optimization methods, since they involve a large number of customers and wide time windows. However, they do not seem to pose any difficulty to Beam-ACO, whereas CA has problems finding the best-known solutions. Since these instances were proposed as larger variants of those in Gendreau et al. [13] (Table 9), this result confirms that Beam-ACO is particularly good in solving this type of instances.

Table 11 provides the results for the symmetric instances proposed by Pesant et al. [10]. We only show results on the test instances. Although both algorithms always find a feasible solution, Beam-ACO always converges to the best-known solution, whereas CA obtains slightly worse solutions in some runs.

Table 12 shows the results obtained by Beam-ACO and CA on the test instances from the benchmark set by Potvin & Bengio [22]. In that table, we also show the results of the current state-of-the-art algorithms for this benchmark set: (1) the results reported by Cheng & Mao [16] for their Ant Colony System algorithm (ACS-TSPTW), and (2) the results of a different Ant Colony System algorithm (labelled ACS-Time) proposed by Gambardella et al. [17]. Both ACS algorithms were exclusively applied to this benchmark set in the original papers. Concerning our results, we can observe that Beam-ACO obtains the best-known feasible solutions in all runs. On the other hand, CA has difficulties finding the best-known solution for seven instances. Despite the fact that Cheng & Mao [16] used a modern computer equipped with an AMD Athlon 1.46 GHz processor (the programming language that was utilized was not

Table 7: Results for asymmetric instances proposed by Ascheuer [24].

		Beam-ACO					CA				
Instance	Best	%inf	RPDm	RPDsd	Tm	Tsd	%inf	RPDm	RPDsd	Tm	Tsd
rbg016b.tw	2094	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg017.2.tw	2351	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg017a.tw	4296	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg019a.tw	2694	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg019d.tw	3479	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg020a.tw	4689	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg021.2.tw	4528	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg021.3.tw	4528	0.00	0.00	0.00	0	0	0.00	0.02	0.04	0	0
rbg021.5.tw	4516	0.00	0.00	0.00	0	0	0.00	0.01	0.04	0	0
rbg021.6.tw	4492	0.00	0.00	0.00	3	3	0.00	0.06	0.03	0	0
rbg021.7.tw	4481	0.00	0.00	0.00	0	0	0.00	0.03	0.05	0	0
rbg021.8.tw	4481	0.00	0.00	0.00	0	0	0.00	0.03	0.06	0	0
rbg021.9.tw	4481	0.00	0.00	0.00	0	0	0.00	0.02	0.04	0	0
rbg027a.tw	5093	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg031a.tw	3498	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg033a.tw	3757	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg034a.tw	3314	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg035a.2.tw	3325	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg035a.tw	3388	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg038a.tw	5699	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg040a.tw	5679	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg041a.tw	3793	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg042a.tw	3260	0.00	0.00	0.00	1	1	0.00	0.00	0.00	1	1
rbg048a.tw	9799	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg049a.tw	13257	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg050b.tw	11957	0.00	0.00	0.00	1	1	0.00	0.00	0.00	0	0
rbg050c.tw	10985	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg055a.tw	6929	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg067a.tw	10331	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg086a.tw	16899	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg125a.tw	14214	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
rbg132.2.tw	18524	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rbg132.tw	18524	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
rbg152.tw	17455	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
rbg172a.tw	17783	0.00	0.00	0.00	18	19	0.00	0.01	0.03	14	17
rbg193.2.tw	21401	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
rbg193.tw	21401	6.67	0.00	0.00	16	14	0.00	0.00	0.00	1	0
rbg201a.tw	21380	0.00	0.00	0.00	8	7	0.00	0.00	0.00	1	0
rbg233.2.tw	26143	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
rbg233.tw	26143	20.00	0.00	0.00	14	9	0.00	0.00	0.00	1	0

Table 8: Results for instances proposed by Dumas et al. [7].

		Beam-ACO					CA				
Benchmark set	Best [12]	%inf	RPDm	RPDsd	Tm	Tsd	%inf	RPDm	RPDsd	Tm	Tsd
n20w20	370.4	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n20w40	342.8	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n20w60	362.0	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n40w20	521.2	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n40w40	512.2	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n40w60	481.4	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n40w80	486.6*	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n40w100	463.0	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n60w20	626.8	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n60w60	672.8*	0.00	0.00	0.00	0	0	0.00	0.00	0.01	0	0
n60w80	628.2	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n60w100	620.2*	0.00	0.00	0.00	0	0	0.00	0.06	0.00	0	0
n80w20	748.2	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n80w60	712.6	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n100w20	823.0	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n100w40	821.0*	0.00	0.00	0.00	0	0	6.67	0.00	0.00	1	0
n100w60	817.2	0.00	0.00	0.00	0	0	1.33	0.00	0.00	1	1
n150w20	978.4	0.00	0.00	0.00	1	1	0.00	0.00	0.00	1	0
n150w40	990.4	0.00	0.00	0.00	4	4	9.33	0.00	0.00	1	0
n150w60	981.4*	0.00	0.73	0.00	4	4	20.00	0.03	0.05	1	1
n200w20	1137.8	0.00	0.00	0.00	7	8	0.00	0.00	0.00	2	0
n200w40	1156.0	8.00	0.00	0.00	15	13	10.67	0.00	0.00	2	0

* New best-known solution found in this paper w.r.t those provided by Carlton & Barnes [12].

mentioned), the computation time of their ACS algorithms is quite high in comparison to Beam-ACO and CA. Furthermore, ACS is clearly inferior to Beam-ACO and CA for what concerns the solution quality. In some cases the ACS algorithms do not even find any feasible solution within five repetitions.

The above results show conclusively that both Beam-ACO and CA are high-performance algorithms for TSPTW with makespan optimization relative to the benchmark sets from the literature. The above results also show that, although CA is able to obtain very good solutions in a very short computation time, Beam-ACO consistently finds the highest quality feasible solutions within the time limit of 60 CPU-seconds. The only methods requiring significant amounts of time (more than 20 seconds) are the ACS algorithms. In most problem instances, both Beam-ACO and CA require less than one second to find the best-known solution. The fact that both Beam-ACO and CA require a mean time of around 20 seconds for a few instances shows that those instances are harder than usual. Nonetheless, given that the time limit was 60 seconds, and that we report the time that the last best-so-far solution was found, a value of 20 seconds indicates that the algorithms converge fast. This may be explained by the fact that we included minimizing time (T_{CPU}) as a criterion for tuning the algorithms (Eq. 8).

Finally, we performed a statistical comparison of Beam-ACO and CA on all test instances, using the best configurations of Beam-ACO and CA for each benchmark set, that is, those ranked first in Tables 5 and 6. We applied Eq. 8 to the results obtained by each run for each instance. This was done in order to take into account both feasibility, quality and computation time. Then, the results are ranked and the Friedman test is applied per benchmark set. The results of the tests are given in Table 13. For all benchmark sets, except for the Ascheuer instances, there is a significant difference in favor of Beam-ACO over CA, which confirms the results observed in the tables above.

6. Summary and Conclusions

This paper presented the adaptation of the Beam-ACO and compressed annealing algorithms from the travel-time variant of the traveling salesman problem with time windows to the variant that considers makespan optimization. This case study supports the intuition that, when tackling a variant of a well-known problem, it might be a good strategy to first adapt the best-known algorithms from the literature to the problem variant, before starting to develop completely new algorithms.

Both Beam-ACO and compressed annealing have been subject to a rigorous tuning process in order to obtain well-working

Table 9: Results for instances proposed by Gendreau et al. [13].

		Beam-ACO					CA				
Benchmark set	Best	%inf	RPDm	RPDsd	Tm	Tsd	%inf	RPDm	RPDsd	Tm	Tsd
n20w120	319.60	0.00	0.00	0.00	0	0	0.00	0.03	0.03	0	0
n20w140	286.20	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n20w160	311.40	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n20w200	281.80	0.00	0.00	0.00	0	0	0.00	0.00	0.02	0	0
n40w120	470.60	0.00	0.00	0.00	0	0	0.00	0.34	0.00	1	0
n40w140	458.20	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n40w180	427.40	0.00	0.00	0.00	0	0	0.00	0.05	0.13	3	3
n60w120	573.80	0.00	0.00	0.00	0	0	0.00	0.48	0.23	2	3
n60w140	600.00	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n60w160	619.60	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n60w180	576.00	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
n60w200	570.20	0.00	0.00	0.00	0	0	0.00	0.04	0.14	4	3
n80w140	672.80	0.00	0.00	0.00	0	0	0.00	0.01	0.03	1	1
n80w160	653.60	0.00	0.00	0.00	1	1	0.00	1.34	0.70	3	6
n80w180	656.40	0.00	0.00	0.00	1	1	0.00	0.24	0.14	3	3
n80w200	646.20	0.00	0.00	0.00	1	0	0.00	0.53	0.42	7	7
n100w80	805.80	0.00	0.00	0.00	0	0	0.00	0.00	0.01	1	0
n100w100	795.80	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n100w120	895.40	0.00	0.00	0.00	0	0	0.00	0.00	0.02	0	0
n100w140	906.40	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n100w160	865.00	0.00	0.00	0.00	0	0	0.00	0.00	0.02	0	0

Table 10: Results for instances proposed by Ohlmann & Thomas [2].

		Beam-ACO					CA				
Instance	Best	%inf	RPDm	RPDsd	Tm	Tsd	%inf	RPDm	RPDsd	Tm	Tsd
n150w120.1	972	0.00	0.00	0.00	1	1	0.00	0.00	0.00	2	1
n150w120.2	917	0.00	0.00	0.00	1	2	0.00	0.00	0.00	2	1
n150w120.4	925	0.00	0.00	0.00	1	1	0.00	0.12	0.47	5	9
n150w120.5	907	0.00	0.00	0.00	1	1	0.00	0.57	0.63	17	18
n150w140.3	844	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n150w140.4	898	0.00	0.00	0.00	1	1	0.00	0.00	0.00	1	0
n150w140.5	926	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n150w160.1	959	0.00	0.00	0.00	1	1	0.00	0.00	0.00	1	0
n150w160.3	934	0.00	0.00	0.00	1	1	0.00	0.00	0.00	8	7
n150w160.5	920	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
n200w120.1	1089	0.00	0.00	0.00	9	8	33.33	3.65	1.22	5	13
n200w120.2	1072	0.00	0.00	0.00	1	2	0.00	0.00	0.00	1	0
n200w120.3	1128	0.00	0.00	0.00	4	5	0.00	0.01	0.02	26	18
n200w120.4	1072	0.00	0.00	0.00	4	5	0.00	0.00	0.00	1	0
n200w120.5	1073	0.00	0.00	0.00	5	5	0.00	0.00	0.00	17	10
n200w140.1	1138	0.00	0.00	0.00	12	12	0.00	0.64	0.72	11	13
n200w140.2	1087	0.00	0.00	0.00	4	5	0.00	0.00	0.00	3	1
n200w140.3	1083	0.00	0.00	0.00	12	11	0.00	0.04	0.08	14	18
n200w140.4	1100	0.00	0.00	0.00	12	8	0.00	0.18	0.00	2	1
n200w140.5	1121	0.00	0.00	0.00	5	9	0.00	0.00	0.00	1	0

Table 11: Results for symmetric instances proposed by Pesant et al. [10].

		Beam-ACO					CA				
Instance	Best	%inf	RPDm	RPDsd	Tm	Tsd	%inf	RPDm	RPDsd	Tm	Tsd
rc201.0	853.71	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc201.1	850.48	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc201.2	883.97	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc201.3	722.43	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc202.0	850.48	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc202.1	702.28	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc202.2	853.71	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc202.3	883.97	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc203.1	850.48	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc204.0	839.24	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc204.1	492.60	0.00	0.00	0.00	0	0	0.00	0.02	0.08	0	0
rc204.2	870.52	0.00	0.00	0.00	1	1	0.00	0.10	0.30	1	0
rc205.0	834.62	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc205.1	899.24	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc205.2	908.79	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc205.3	684.21	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0
rc206.1	756.45	0.00	0.00	0.00	0	0	0.00	0.00	0.00	1	0
rc206.2	776.19	0.00	0.00	0.00	3	3	0.00	0.29	0.24	2	2
rc207.1	785.37	0.00	0.00	0.00	0	0	0.00	0.32	0.25	0	0
rc208.0	836.04	0.00	0.00	0.00	18	15	0.00	0.34	0.27	0	0
rc208.1	615.51	0.00	0.00	0.00	14	12	0.00	0.10	0.14	0	0
rc208.2	596.21	0.00	0.00	0.00	1	1	0.00	0.19	0.39	0	0

Table 12: Results for instances from Potvin & Bengio [22].

		Beam-ACO					CA					ACS-TSPTW [16]		ACS-Time [16]	
Instance	Best	%inf	RPDm	RPDsd	Tm	Tsd	%inf	RPDm	RPDsd	Tm	Tsd	RPDm	Tm	RPDm	Tm
rc201.1	592.06	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	0.00	101	0.00	97
rc201.2	860.17	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	2.01	246	0.74	263
rc201.4	889.18	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	1.28	151	*inf.*	*inf.*
rc202.1	850.48	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	3.56	242	3.56	242
rc202.2	338.52	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	0.00	47	12.98	47
rc202.3	894.10	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	-0.21(!)	190	*inf.*	*inf.*
rc203.1	488.42	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	37.81	79	22.98	80
rc203.2	853.71	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	8.56	256	6.75	279
rc203.3	921.44	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	*inf.*	*inf.*	*inf.*	*inf.*
rc204.1	917.83	0.00	0.00	0.00	27	21	0.00	1.35	0.94	14	15	3.47	438	*inf.*	*inf.*
rc204.3	455.03	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	41.10	127	17.19	129
rc205.1	417.81	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	1.06	47	0.90	47
rc205.2	820.19	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	0.00	181	0.00	195
rc205.3	950.05	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	0.00	275	0.12	273
rc205.4	837.71	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	3.91	187	1.39	180
rc206.1	117.85	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	0.00	13	0.00	13
rc206.2	870.49	0.00	0.00	0.00	0	0	0.00	0.89	0.68	1	0	5.11	306	4.19	304
rc207.1	804.67	0.00	0.00	0.00	1	1	0.00	0.74	0.31	1	0	7.00	258	10.52	258
rc207.2	713.90	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	*inf.*	*inf.*	10.99	*inf.*
rc207.3	745.77	0.00	0.00	0.00	3	4	0.00	0.65	0.46	0	0	28.15	242	13.30	234
rc207.4	133.14	0.00	0.00	0.00	0	0	0.00	0.00	0.00	0	0	0.00	22	0.00	23
rc208.1	810.70	0.00	0.00	0.00	4	4	0.00	0.63	0.75	0	0	15.31	335	11.21	332
rc208.2	579.51	0.00	0.00	0.00	9	8	0.00	0.31	0.02	0	0	24.63	186	5.06	185
rc208.3	686.80	0.00	0.00	0.00	1	1	0.00	1.32	1.56	0	0	15.76	292	7.68	296

(!) The results reported by Cheng & Mao [16] are better than the best value we ever found. We suspect a typo in the original paper.

inf. The algorithm did not find a single feasible solution over 5 independent runs.

Table 13: Friedman test of Beam-ACO vs. CA configurations per instance set (see text for details).

Benchmark	ΔR_α	**Configuration** (ΔR)
Ascheuer	∞	*Beam-ACO (0), CA (6)*
Dumas et al.	5.37	*Beam-ACO (0),* CA (6)
Gendreau et al.	4.76	*Beam-ACO (0),* CA (13)
Ohlmann & Thomas	7.11	*Beam-ACO (0),* CA (11)
Pesant et al.	5.37	*Beam-ACO (0),* CA (6)
Potvin & Bengio	4.71	*Beam-ACO (0),* CA (7)

parameter settings. Moreover, a comprehensive experimental analysis has shown that both Beam-ACO and compressed annealing perform significantly better than algorithms recently proposed specifically for makespan optimization. Concerning the comparison between Beam-ACO and compressed annealing, a slight but consistent advantage of Beam-ACO over CA was observed.

Future work might consist in adapting other algorithms originally proposed for the travel-time minimization to makespan minimization. However, this may prove to be difficult in the case of algorithms that rely on heuristics specific to the travel-time objective. Moreover, we are not aware of other state-of-the-art metaheuristics for the travel-time objective that are candidates to perform well for the makespan objective [3].

Another interesting extension of the present work concerns the automatic development of algorithms for less known problem variants on the basis of well-performing algorithms for well known (and closely related) problems by means of automatic configuration tools [27]. The resulting algorithms would then be benchmarked against the current state of the art for those less known problem variants in the same way as done in this paper.

References

[1] M. W. P. Savelsbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1) (1985) 285–305, doi: 10.1007/BF02022044.

[2] J. W. Ohlmann, B. W. Thomas, A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows, INFORMS Journal on Computing 19 (1) (2007) 80–90, ISSN 1526-5528, doi: 10.1287/ijoc.1050.0145.

[3] M. López-Ibáñez, C. Blum, Beam-ACO for the travelling salesman problem with time windows, Computers & Operations Research 37 (9) (2010) 1570–1583, doi:10.1016/j.cor.2009.11.015.

[4] N. Christofides, A. Mingozzi, P. Toth, State-space relaxation procedures for the computation of bounds to routing problems, Networks 11 (2) (1981) 145–164, doi:10.1002/net.3230110207.

[5] E. K. Baker, An Exact Algorithm for the Time-Constrained Traveling Salesman Problem, Operations Research 31 (5) (1983) 938–945, doi: 10.1287/opre.31.5.938.

[6] A. Langevin, M. Desrochers, J. Desrosiers, S. Gélinas, F. Soumis, A Two-Commodity Flow Formulation for the Traveling Salesman and Makespan Problems with Time Windows, Networks 23 (7) (1993) 631–640.

[7] Y. Dumas, J. Desrosiers, E. Gelinas, M. M. Solomon, An Optimal Algorithm for the Traveling Salesman Problem with Time Windows, Operations Research 43 (2) (1995) 367–371.

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

[9] E. Balas, N. Simonetti, Linear Time Dynamic-Programming Algorithms for New Classes of Restricted TSPs: A Computational Study, INFORMS Journal on Computing 13 (1) (2001) 56–75, doi: 10.1287/ijoc.13.1.56.9748.

[10] G. Pesant, M. Gendreau, J.-Y. Potvin, J.-M. Rousseau, An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows, Transportation Science 32 (1998) 12–29.

[11] F. Focacci, A. Lodi, M. Milano, A Hybrid Exact Algorithm for the TSPTW, INFORMS Journal on Computing 14 (2002) 403–417.

[12] W. B. Carlton, J. W. Barnes, Solving the traveling-salesman problem with time windows using tabu search, IIE Transactions 28 (1996) 617–629.

[13] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows, Operations Research 46 (1998) 330–335.

[14] R. Wolfler Calvo, A New Heuristic for the Traveling Salesman Problem with Time Windows, Transportation Science 34 (1) (2000) 113–124, doi: 10.1287/trsc.34.1.113.12284.

[15] R. Ferreira da Silva, S. Urrutia, A general VNS heuristic for the traveling salesman problem with time windows, Discrete Optimization 7 (4) (2010) 203–211.

[16] C.-B. Cheng, C.-P. Mao, A modified ant colony system for solving the travelling salesman problem with time windows, Mathematical and Computer Modelling 46 (2007) 1225–1235, doi:10.1016/j.mcm.2006.11.035.

[17] L. M. Gambardella, É. D. Taillard, G. Agazzi, MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw Hill, London, UK, 63–76, 1999.

[18] D. Favaretto, E. Moretti, P. Pellegrini, Ant colony system approach for variants of the traveling salesman problem with time windows, Journal of Information and Optimization Sciences 27 (1) (2006) 35–54.

[19] C. Blum, M. Mastrolilli, Using Branch & Bound Concepts in Construction-Based Metaheuristics: Exploiting the Dual Problem Knowledge, in: T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, vol. 4771 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, 123–139, 2007.

[20] H. Juillé, J. B. Pollack, A Sampling-Based Heuristic for Tree Search Applied to Grammar Induction, in: Proceedings of AAAI 1998 – Fifteenth National Conference on Artificial Intelligence, MIT Press, Cambridge, MA, 776–783, 1998.

[21] W. Ruml, Incomplete Tree Search using Adaptive Probing, in: Proceedings of IJCAI 2001 – Seventeenth International Joint Conference on Artificial Intelligence, IEEE Press, 235–241, 2001.

[22] J.-Y. Potvin, S. Bengio, The Vehicle Routing Problem with Time Windows Part II: Genetic Search, INFORMS Journal on Computing 8 (1996) 165–172.

[23] M. M. Solomon, Algorithms for the Vehicle Routing and Scheduling Problems with Time Windows, Operations Research 35 (1987) 254–265.

[24] N. Ascheuer, Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems, Ph.D. thesis, Technische Universität Berlin, Berlin, Germany, 1995.

[25] P. Balaprakash, M. Birattari, T. Stützle, Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement, in: T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, vol. 4771 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, 108–122, 2007.

[26] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, Iterated Race for Automatic Algorithm Configuration, Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`, 2011.

[27] M. López-Ibáñez, T. Stützle, The Automatic Design of Multi-Objective Ant Colony Optimization Algorithms, IEEE Transactions on Evolutionary Computation 16 (6) (2012) 861–875, doi: 10.1109/TEVC.2011.2182651.