Information Sciences 432 (2018) 231–244

Contents lists available at ScienceDirect

Information Sciences

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

Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing

Mohamed El Yafrani*, Belaïd Ahiod

LRIT, CNRST (URAC 29), Faculty of Science, Mohammed V University in Rabat, Rabat BP 1014, Morocco

A R T I C L E   I N F O

Article history:
Received 5 June 2017
Revised 6 December 2017
Accepted 7 December 2017
Available online 9 December 2017

Keywords:
Interdependence
Combinatorial optimization
Large-scale optimization
Traveling Thief Problem
Simulated annealing
Local search

A B S T R A C T

Many real-world problems are composed of multiple interacting sub-problems. However, few investigations have been carried out to look into tackling problems from a metaheuristics perspective. The Traveling Thief Problem (TTP) is a new NP-hard problem with two interdependent components that aim to provide a benchmark model to better represent this category of problems. In this paper, TTP is investigated theoretically and empirically. Two algorithms based on a 2-OPT steepest ascent hill climbing algorithm and the simulated annealing metaheuristic named *CS2SA** and *CS2SA-R* are proposed to solve the problem. The obtained results show that the proposed algorithms are efficient for many TTP instances of different sizes and properties and are very competitive in comparison with two of the best-known state-of-the-art algorithms.

1. Introduction

Tackling real-world optimization problems is often more complex than solving the Traveling Salesman Problem or finding the chromatic number of a graph. Most real-world problems are constrained and often made of multiple sub-problems which on their own are hard to solve. These problems have the following properties:

- *Composition:* The overall problem is composed of multiple sub-problems, such as each sub-problem being hard to solve on its own.
- *Interdependence:* The sub-problems are related to each other. Therefore, they cannot be solved in isolation.

These kinds of problems are widely found in supply chain management. Indeed, in supply chain optimization, all operations are often complex problems that interact with each other (distribution, scheduling, loading, transportation, etc.) [5,22,23]. Many other multi-component problems exist in various fields. Examples include the traffic grooming, routing and wavelength assignment problems in computer networks [21], the routing and placement problem in VLSI printed circuit board design [4], the routing problems combined with loading/unloading operations [24,34,38], and the logistics problem of optimizing ground movement in airports [42].

During the last decades, the field of metaheuristics and evolutionary computation have made great progress. Many complex problems are nowadays easily solvable using modern heuristic approaches. However, many researchers focus on benchmark problems that do not efficiently reflect the composite nature of real-world problems.

* Corresponding author.
E-mail address: melyafrani@acm.org (M. El Yafrani).

https://doi.org/10.1016/j.ins.2017.12.011
0020-0255/© 2017 Elsevier Inc. All rights reserved.

The Traveling Thief Problem (TTP) is a recently introduced benchmark problem that aims to provide testbeds for solving multi-components problems with interdependence [4]. The simplified version of the problem is illustrated in the following question:

Given n cities and m items scattered among these cities, a thief with his rented knapsack should visit all n cities, once and only once per each, pick up some items, and return to the first city. The heavier the knapsack gets, the slower the thief becomes. What is the best path and picking plan to adopt in order to achieve the best benefits?

It is clear, given the above statement, that the TTP has two components, namely the Traveling Salesman Problem (TSP) and the 0/1-Knapsack Problem (KP). Moreover, these two components are interdependent since the weight of the knapsack influences the speed of the thief.

Since the introduction of the TTP, several algorithms were proposed to solve it. An Evolutionary Algorithm and a Random Local Search were introduced by Polyakovskiy et al. [36] as a first attempt to solve the problem using simple heuristic algorithms.

An approach named CoSolver was introduced by Bonyadi et al. [6] to tackle the TTP using decomposition. The framework consists of separating and processing the two sub-problems while maintaining a communication between them.

Two evolutionary algorithms were proposed by Mei et al. [30] to solve an early version of the TTP[1] The first approach is a Cooperative Coevolution algorithm that solves the sub-problems in isolation without taking the interdependence into consideration. The second is a memetic algorithm that solves the problem as a whole. The authors empirically show the importance of considering the dependencies in order to efficiently solve the overall problem.

Mei et al. [29] introduced a memetic algorithm called *MATLS* and multiple complexity reduction techniques in order to solve very large instances in a time window of 10 min.

Faulkner et al. [16] investigated multiple operators and heuristics and did a comprehensive comparison with existing approaches. They proposed a greedy picking approach and multiple simple and complex heuristics. The work mainly focuses on optimizing the picking plan when given a particular tour generated using the Lin–Kernighan heuristic [26]. Additionally, in an attempt to avoid the bias toward the KP component, the authors proposed an insertion operator for iteratively optimizing the tour for a particular picking plan. The proposed heuristics are the result of combining all these operators and routines. According to the obtained results, the best performing heuristic on average is a simple iterative heuristic called *S5*. The heuristic focuses on optimizing the picking plan given a fixed tour generated using the Lin–Kernighan heuristic.

El Yafrani and Ahiod [14] introduced two heuristics and proposed an empirical study in order to compare two types of search heuristics: population-based and single-solution algorithms. The first proposed heuristic is called *MA2B*, a memetic algorithm based on fast local search heuristics. The second approach is called *CS2SA*, a two-part search heuristic combining a 2-OPT steepest ascent hill climbing heuristic and an adapted simulated annealing for efficient item picking. The two proposed heuristics were compared to *MATLS* and *S5* and were shown to be very competitive.

Wagner [40] recently investigated the idea of focusing less on short TSP tours and more on good TTP tours, which can be of a longer distance. The proposed algorithms are shown to be efficient for small instances with up to 250 cities and 2000 items. However, their performance decreases significantly for larger instances.

El Yafrani and Ahiod [15] focused on designing TTP-specific local search approaches based on neighborhood operators. Their paper discusses the advantages and the drawbacks of such a solution. The reported results show that this approach was competitive against a basic Evolutionary Algorithm (*EA*) and a Radomized Local Search (*RLS*) for different small and midsize instances.

Other researchers took a different path by exclusively investigating the KP component of the TTP [35]. This problem is referred to as the Unconstrained Nonlinear Knapsack Problem or the Packing While Traveling Problem. Papers focusing on this problem introduced an exact solver [34], proposed a fully polynomial-time approximation scheme [32], and investigated the effect of the renting rate parameter [43].

Interested readers can also refer to the work of Wagner et al. [41] who investigated multiple TTP state-of-the-art heuristics in order to establish the strengths and weaknesses of these approaches. These algorithms are then used to construct TTP algorithm portfolios.

In this paper, we follow the ideas introduced in [6,29] to propose further improvements and introduce a heuristic idea based on CoSolver able to efficiently solve TTP instances of various sizes within a decent time budget. The CoSolver framework proposes the idea of dealing with the sub-problems separately. Although the proposed CoSolver model can be used in a parallel fashion (using separate resources to process each sub-problem), our implementation adopts a sequential structure but considers the same decomposition technique. Our approach combines the 2-OPT local search heuristic [8] for the TSP component and the simulated annealing metaheuristic [1] for the KP component.

The rest of this paper is organized as follows: in Section 2, the TTP is formally defined and briefly investigated. Section 3 is dedicated to introducing our approach to solving the TTP. The tests and results are reported and discussed in Section 4. Section 5 concludes the paper and outlines some future directions.

[1] This version is different from the TTP as redefined by Polyakovskiy et al. [36] in which an item can be found in only one city.

2. Background

In this section, we present some background information about the TTP. The problem is formulated, the interdependence is analyzed, and the CoSolver framework is presented in a simplified manner.

2.1. The Traveling Thief Problem

Herein, we formulate the TTP which combines two other well-known benchmark problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). In the TTP, we consider n cities and the associated distance matrix $\{d_{ij}\}$ such that $i, j \in \{1, \dots, n\}$. We suppose there are m items scattered in these cities, each item k having a profit p_k and a weight w_k. A thief with his rented knapsack is going to visit all these cities once while picking up some items to fill his knapsack. We note W the maximum capacity of the knapsack and $v_{\min}$ and $v_{\max}$ denote the minimum and maximum possible velocity respectively. Additionally, we consider the following constraints and parameters:

- Each item is available in only one city. We note $\{A_i\}$ as the availability vector such as $A_i \in \{1, \dots, n\}$. $\{A_i\}$ contains the reference to the city that contains the item i.
- We suppose that the knapsack is rented, and we note R as the renting price per time unit.
- The velocity of the thief changes accordingly to the knapsack weight. The heavier the knapsack gets, the slower the thief becomes. v_{x_i} denotes the velocity of the thief at city x_i as defined in Eq. (1).

$$v_{x_i} = v_{max} - C \times w_{x_i} \tag{1}$$

where $C = \frac{v_{\max} - v_{\min}}{W}$ is a constant value and w_{x_i} represents the weight of the knapsack at city x_i.

The goal of the problem is to find the tour x and the picking plan z that maximize the total travel gain G defined in Eq. (2).

$$G(x, z) = g(z) - R \times f(x, z) \tag{2}$$

where $g(z) = \sum_{k=1}^{m} p_k \times z_k$ is the total value of the items subject to $\sum_{k=1}^{m} w_k \times z_k \le W$, and $f(x, z) = \sum_{i=1}^{n-1} t_{x_i, x_{i+1}} + t_{x_n, x_1}$ is the total travel time such that $t_{x_i, x_{i+1}} = \frac{d_{x_i, x_{i+1}}}{v_{x_i}}$ is the travel time from x_i to x_{i+1}.

A TTP solution is represented as follows:

- The tour $x = (x_1, \dots, x_n)$ is a vector containing the ordered list of cities.
- The picking plan $z = (z_1, \dots, z_m)$ is a binary vector such as z_i is equal to 1 if the item i is picked, 0 otherwise.

The NP-hardness of the TTP is trivial as it can be seen as a generalization of the TSP or the KP, both known to be NP-hard [28,33]. The TTP can be reduced to a TSP by assigning a zero to all the profits. Similarly, it can be seen as a KP if one city is uniquely considered or if the renting rate is set to zero. Polyakovskiy and Neumann [35] went even further and proved the NP-hardness of the unconstrained KP component, which is a version that does not consider a knapsack capacity limit.

2.2. The interdependence in TTP

The interdependence between the KP and the TSP has already been thoroughly investigated in [4,30]. Thus, it has been shown that optimizing the sub-problems in separation does not guarantee the finding of good solutions for the overall problem. Therefore, finding good global solutions requires an algorithm that takes the interdependence of components into consideration, which makes the design of such an algorithm quite difficult.

In this subsection, the interdependence in the TTP is analyzed through its effect on solving the problem given a particular operator.

In the TSP component

The first consequence of the interdependence of the TTP relies on the fact that the TSP component stops being a symmetric problem. This is due to the metric change in the TTP, which considers the time of travel instead of the distance. The travel time from a city A to another city B is not equal to the travel time from B to A. Indeed, the cities may contain items of different weights which leads to different velocities.

Therefore, when using the 2-OPT heuristic, retrieving the objective value from a tour in which two arcs have been exchanged has a linear complexity $O(N)$, such as N is the number of cities in the sub-tour, unlike the symmetric TSP, in which the retrieval of the objective value for neighbor solutions can be performed instantly ($O(1)$).

In the TTP, the time of travel changes according to the weight of the knapsack. This dynamic aspect of the TTP makes it even harder to tackle the problem. However, by keeping track of the time and weight at each city, it is possible to recover the objective value of a 2-OPT neighbor in order to avoid using the more expensive objective function.

Consider a TTP solution (x, z) such as $x = x_1, \ldots, x_n$ is the tour and $z = z_1, \ldots, z_m$ is the picking plan. Let $f(x, z)$ be the total traveling time defined in Eq. (3).

$$\begin{aligned} f(x,z) = & \frac{d_{x_1,x_2}}{v_{max} - C \times w^{acc}_{x_1}} + \ldots \\ & + \frac{d_{x_{i-1},x_i}}{v_{max} - C \times w^{acc}_{x_{i-1}}} + \frac{d_{x_i,x_{i+1}}}{v_{max} - C \times w^{acc}_{x_i}} + \frac{d_{x_{i+1},x_{i+2}}}{v_{max} - C \times w^{acc}_{x_{i+1}}} \\ & + \cdots + \frac{d_{x_n,x_1}}{v_{max} - C \times w^{acc}_{x_n}} \end{aligned} \tag{3}$$

where w^{acc}_a denotes the current weight at city a.

Let (x^{ij}, z) be the neighbor solution of (x, z) such as the 2-OPT operator is applied at the indices i and j. Computing the total traveling time for (x^{ij}, z) can be done using Eq. (4).

$$\begin{aligned} f(x^{ij},z) = & \frac{d_{x_1,x_2}}{v_{max} - C \times w^{acc}_{x_1}} + \ldots \\ & + \frac{d_{x_i,x_j}}{v_{max} - C \times w^{acc}_{x_i}} + \frac{d_{x_j,x_{j-1}}}{v_{max} - C \times (w^{acc}_{x_i} + w^{reg}_{x_j})} + \frac{d_{x_{j-1},x_{j-2}}}{v_{max} - C \times (w^{acc}_{x_i} + w^{reg}_{x_j} + w^{reg}_{x_{j-1}})} + \ldots \\ & + \frac{d_{x_{i+1},x_{j+1}}}{v_{max} - C \times (w^{acc}_{x_{i+1}} + w^{reg}_{x_j} + w^{reg}_{x_{j-1}} + \cdots + w^{reg}_{x_{i+1}})} \\ & + \frac{d_{x_{j+1},x_{j+2}}}{v_{max} - C \times w^{acc}_{x_{j+1}}} + \cdots + \frac{d_{x_n,x_1}}{v_{max} - C \times w^{acc}_{x_n}} \end{aligned} \tag{4}$$

where w^{reg}_a represents the weight of all the items taken from city a.

In the KP component

The second consequence of interdependence occurs when picking or unpacking items (i.e. using the bit-flip operation). In fact, it is impossible to recover the traveling time in a constant time since the change affects all the terms that come after the region of change (i.e. the term corresponding to the city from which the item was picked or unpacked). Nevertheless, the objective value of a neighbor picking plan can be recovered by reusing the terms before the region of change and keeping track of time and the weight of the original solution.

This technique has a linear complexity $O(N)$, such as N is the number of cities from the location where the bit-flip happens to the last city in the tour.

In this second case, we consider the solution (x, z^k) such as the item z_k is picked (or unpacked) from city i.

The value $f(x, z^k)$ can be recovered partially from $f(x, z)$ by reusing the terms before the region of change, and keeping track of time and weight of the original solution.

$$\begin{aligned} f(x,z^k) = & \frac{d_{x_1,x_2}}{v_{max} - C \times w_{x_1}} + \ldots \\ & + \frac{d_{x_{i-1},x_i}}{v_{max} - C \times w_{x_{i-1}}} + \frac{d_{x_i,x_{i+1}}}{v_{max} - C \times (w_{x_i} + \Delta_W)} + \frac{d_{x_{i+1},x_{i+2}}}{v_{max} - C \times (w_{x_{i+1}} + \Delta_W)} \\ & + \cdots + \frac{d_{x_n,x_1}}{v_{max} - C \times (w_{x_n} + \Delta_W)} \end{aligned} \tag{5}$$

where Δ_W represents the difference of weight after applying the bit-flip operator.

Composition vs. interdependence

In the TTP, the combination of TSP and KP increases the number of decision variables which make the problem harder than regular benchmark problems. However, the interdependence makes the problem even harder to crack by increasing the complexity of evaluation as shown above. Thus, designing a strong[2] algorithm that intelligently deals with the TTP's computational challenges requires more effort.

2.3. CoSolver

CoSolver is a framework proposed by Bonyadi et al. [6] to solve the Traveling Thief Problem. The idea behind the algorithm is simple: decompose the overall problem and solve components separately using a fitness function that takes into consideration the interdependence between the two components.

Thus, given an initial solution (x_0, z_0), the TTP is decomposed into the following problems:

[2] A strong algorithm, as defined in [31], makes strong assumptions about the problem domain and exploits these assumptions to solve the problem.

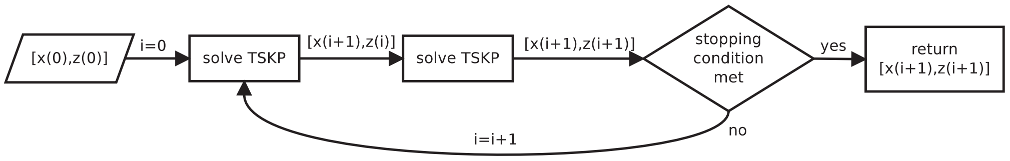

Fig. 1. A simplified CoSolver framework for the TTP.

- *The Traveling Salesman with Knapsack Problem (TSKP):* consists of finding the best tour x' that, combined with the last found picking plan z', optimizes the TTP objective function G. Also, because the total profit function g does not depend on the tour, instead of maximizing G, we can consider minimizing the total travel cost T[3] (see Eq. (6)).

$$T(x,z) = R \times f(x,z) \tag{6}$$

- *The Knapsack on the Route Problem (KRP):* consists of finding the best picking plan z' that maximizes the total gain G when combined with the last found tour x'. Note that this problem is similar to the Unconstrained Nonlinear Knapsack Problem briefly mentioned in the introduction section.

Fig. 1represents a flowchart of a simplified version of the CoSolver framework.

3. Proposed approach

In this section, we propose a solution initialization strategy and present our approach to solving the TTP using a two-stage algorithm.

3.1. Solution initialization

In the TTP, the initial solution has a big impact on the search algorithm – especially the initial tour. In fact, most current algorithms use a sophisticated heuristic to generate an initial tour (Lin–Kernighan heuristic, Boruvka algorithm, 2-opt, etc.). Thus, the initialization strategy should be chosen carefully. In our implementation, we also use a heuristic approach to initialize both the tour and the picking plan. Firstly, the initial tour is generated using the Concorde's implementation of the Lin–Kernighan heuristic [2,26]. Then, the picking plan is initialized using the following approach:

1. The insertion heuristic proposed by Mei et al. [29] is used to fill the knapsack with items. This technique uses three fitness approximations to select the most profitable items.
2. A simple bit-flip search is then applied on inserted items to eliminate some useless items from the picking plan using the objective function.

In the rest of this paper, we will refer to this heuristic as the *insertion & elimination heuristic*.

3.2. Performance enhancement techniques

Our algorithm uses the following complexity reduction and performance enhancement techniques.

TSKP neighborhood reduction

The Delaunay triangulation [9] is used as a candidate generator for the 2-OPT heuristic. This technique is also used in the memetic algorithm from Mei et al. [29]. Using the Delaunay triangulation to generate candidates reduces the time complexity without significantly decreasing the quality of solutions. In a Delaunay graph, each city has only 6 surrounding cities on average. Thus, the time complexity of the 2-OPT heuristic becomes $O(n)$ instead of $O(n^2)$.

In our implementation, we use the worst-case optimal divide-and-conquer Delaunay triangulation algorithm [20] which has a time complexity of $O(n * log(n))$.[4]

Objective value recovery

Instead of using the expensive objective function to evaluate neighbors, the objective value can be recovered more quickly by keeping track of time and weight information at each city of a given tour. The following history vectors are used to perform this operation.

- *Time accumulator (t^{acc}):* a vector that contains the current time at each city of the tour.
- *Weight accumulator (w^{acc}):* a vector that contains the current weight at each city of the tour.

[3] Minimizing the total travel time can also be considered.

[4] The source codes of the algorithm are found in the website http://www.cs.rmit.edu.au/~gl/delaunay.html.

- *Time register (t^{reg}):* a vector that contains the added time at each city of the tour.
- *Weight register (w^{reg}):* a vector that contains the added weight at each city of the tour.

Note that these vectors need to be updated at each iteration of the search heuristic. A similar technique named *incremental evaluation* was proposed in [29]. In our implementation, we go further and use these techniques to recover the vectors themselves. This is quite useful, especially when the problem has a large dimension and the algorithm requires many iterations (e.g. the first improvement hill climbing heuristic) or uses many calls of the evaluation function and changes constantly the solution (e.g. the simulated annealing in Algorithm 2).

Very large-scale instances

For very large instances, local search heuristics such as 2-OPT and the bit-flip search take too long and make very little improvement. Therefore, in order to avoid wasting runtime on small improvements, we use the following techniques:

- *2-OPT with early break:* A threshold acceptance is utilized in our 2-OPT heuristic. Therefore, an improved neighbor solution s' of s is accepted only if the improvement is important (surpasses a threshold T): $| G(s') - G(s) | > T$.
- *Insertion without elimination:* The bit-flip search is skipped for very large instances in the *insertion & elimination heuristic*. Therefore, the elimination task is left to the KRP solver.

3.3. Proposed approach

The proposed heuristic is a single solution algorithm that implements the CoSolver framework. In addition to the techniques presented above, we use the 2-OPT heuristic [8] to solve the TSKP component and the simulated annealing heuristic [1] to solve the KRP component. The algorithm will be referred to as *CS2SA* (which stands for CoSolver-based with 2-OPT and Simulated Annealing).

The algorithm starts with an initial tour generated using the chained Lin–Kernighan heuristic. The resulting tour is then passed to the *insertion & elimination heuristic* which generates a decent initial picking plan. The obtained tour and picking plan are then combined and given to the TSKP solver which improves the current tour accordingly to the picking plan using the 2-OPT heuristic (see Algorithm 1). Then, the obtained tour is combined with the current picking plan and passed to the KRP solver which uses the simulated annealing metaheuristic to improve the picking plan (see Algorithm 2). The obtained picking plan is combined with the current tour, and, if no improvements are made, the best-found solution is returned. Otherwise, the tour and picking plan are passed again to the TSKP solver.

Algorithm 1 2-OPT heuristic search for TSKP (best improvement version).

```
1  $s \leftarrow$ starting solution
2  $f \leftarrow$ starting travel time
3  $T \leftarrow$ initialize the threshold parameter
4  repeat
5      $improved \leftarrow false$
6      for $\bar{s}$ in $\mathcal{N}(s)$ do                                ▷ browse 2-OPT neighborhood
7          $\bar{f} \leftarrow$ evaluate $\bar{s}$ using objective value recovery technique
8          if $\bar{f} - f < T$ then                                    ▷ acceptance condition
9              $i_{best} = i$
10             $j_{best} = j$
11             $f = \bar{f}$
12             $improved = true$
13         end if
14     end for
15     if $improved$ then
16         $s \leftarrow$ apply 2-OPT exchange on $s$ at $i_{best}$ and $j_{best}$
17         recover history vectors
18     end if
19 until not $improved$
```

TSKP solver: The 2-OPT steepest ascent hill climbing heuristic

The 2-OPT heuristic is described in Algorithm 1. The algorithm generates a neighborhood noted $\mathcal{N}$ at each iteration (line 6). The generated neighborhood uses the Delaunay triangulation as explained in the previous sub-section. Then, the objective value recovery technique is used to evaluate neighbors (line 7). The indices of a neighbor and the travel time are stored when a better objective value is found (lines 8–13). At the end of each iteration, if a better solution was found during the neighborhood search, the current solution will be updated and the history vectors will be recovered. The neighborhood search continues until no improvement is made (lines 15–18).

Algorithm 2 Simulated annealing for KRP.

```
s ← starting solution
s_best ← s
G ← starting gain
p ← starting profit
f ← starting travel time
T ← T_0                                                        ▷ initialize the temperature parameter
repeat
    for u in nb_trials do
        k ← pick an item randomly
        $\overline{p} \leftarrow p + p_k$
        if $\overline{p}$ > knapsack capacity then skip this iteration end if
        $\overline{f}$ ← evaluate time using the objective value recovery technique
        $\overline{G} \leftarrow \overline{p} - R * \overline{f}$
        $\mu$ ← random number between 0 and 1
        $energy\_gap \leftarrow \overline{G} - G$
        if $energy\_gap > 0$ or $exp(energy\_gap/T) > \mu$ then          ▷ Boltzmann condition
            $G \leftarrow \overline{G}$
            $p \leftarrow \overline{p}$
            $f \leftarrow \overline{f}$
            s ← apply bit flip at k
            recover history vectors
        end if
    end for
    if improvement made then $s_{best} \leftarrow s$ end if
    $T \leftarrow \alpha * T$                                          ▷ cool down temperature
until $T > T_{abs}$                                                 ▷ absolute temperature reached
```

Note that we use the best improvement strategy, which implies that the entire neighborhood is searched at each iteration. Also, the threshold parameter T is set to 0 for small and mid-size instances and -10 for very large instances.

KRP solver: The simulated annealing

To tackle the KRP sub-problem, we adapt the simulated annealing metaheuristic as shown in Algorithm 2. The algorithm uses a random candidate generator (line 9). Therefore, an internal loop is used to explore more possibilities (line 8). The mutated solution is evaluated using the objective value recovery technique (line 12). The acceptance test is based on the Boltzmann condition (lines 14–16). If the mutated solution is accepted, the current parameters and solution are updated and the history vectors are recovered (lines 17–21). If a better solution is found, the best-found solution is updated (line 24). At the end of each iteration, the temperature is cooled down (line 25). The algorithm stops when the absolute temperature is reached (line 26).

In our implementation, we use the following parameters:

- T_{abs}: the absolute temperature, set to 1.
- T_0: the initial temperature.
- α: the temperature cooling parameter.
- *nb_trials*: number of repetitions per iteration. This parameter should be selected more carefully. A high number of trials explores more possibilities but also makes the metaheuristic slow, while a low number of trials explores fewer possibilities but makes the algorithm faster. The value of this parameter is chosen according to the number of items.

More details on tuning these parameters are presented in the next section.

3.4. CS2SA variants

In this paper, two versions of the *CS2SA* algorithm are proposed. The first is called *CS2SA**, a straight implementation of *CS2SA* for which the parameters are optimized using the advanced tuning strategy described in Section 4.

The second variant is called *CS2SA-R*, which uses a random restart instead of directly returning the best found solution when no improvements are made. Instead of using a stopping criterion based on improvement existence, *CS2SA-R* uses the entire available time budget.

Table 1
A comparison of the impact of component solvers on the objective function (G), the travel time (f), and the final profit (g) for the instance *eil51_n150_bounded-strongly-corr_01*.

	Initial solution	After TSP hill climbing	After KP simulated annealing	After KP hill climbing
G	−779	283	6392	6101
f	1070	1019	629	642
g	21,247	21,247	19,343	19,325

4. Experimental study

In this section, we start by studying the impact of the two component solvers on the overall objective. Then, we present the approach used to tune the simulated annealing parameters. Afterwards, the proposed algorithms are compared against two of the best-known TTP algorithms, namely *MATLS* and *S5*.

4.1. On the impact of component solvers

Most researchers consider the KP component as the most critical part of the TTP and mainly focused on it to design solvers. We believe that this assumption is fairly justified. However, none of the TTP related papers we are aware of discussed or investigated this aspect. Performing multiple experiments on the TTP shows that the KP component provides more room for optimization compared to the TSP component. Herein, we experimentally show that the total gain (G) increases more significantly when the KP part is tackled compared to when the TSP part is optimized.

To perform our study, we start by generating an initial solution combining a near-optimal tour generated using the Lin–Kernighan heuristic, and an optimal picking plan generated using dynamic programming. Afterwards, a component solvers is applied individually to solve the problem. We are interested in the rate of improvement brought by each component solver. Table 1summarizes the results of the experiment. We report the objective value (second row), the travel time (third row), and the final profit (fourth row) for the initial solution (second column), the solution after applying the hill climbing to the TSP component (third column), and the solution after applying the simulated annealing to the KP component (fourth column). Additionally, for the sake of making a fair comparison, we also report the solution after applying a simple hill climber [13] to the KP component (fifth column). Note that for simplicity reasons, the results are reported for only one instance. Nevertheless, all the other tests we have performed for other instances show the same behaviour.

Clearly, improving the picking plan has more impact on the overall objective than improving the tour. The results tend to show that this behaviour is not due the performance of the component solvers, but instead to the problem nature itself. Eq. (2) provides a simple explanation: *both the final profit and the travel time depend on the picking plan, while the tour is only present in the travel time's function. Therfore, optimizing the picking plan has more impact on the overall objective.*

It is worth noting that a second version of the TTP was proposed in the original paper named TTP2 [4]. In TTP2, in addition to the speed variation constraint (Eq. (1)), it is also supposed that the value of items drops with time. We believe that this would be a more balanced model due to the fact that both the final profit (g) and the travel time (f) are dependent on the picking plan and the tour. This also constitutes a more challenging problem as it would be difficult to develop speedup approaches to improve the runtime.

4.2. Parameters tuning

In this first set of experiments, we investigate the simulating annealing's parameter tuning. Indeed, one of the hardest tasks when implementing the simulating annealing metaheuristic is to find the best values for its parameters. Thereby, the best-known way to find a good setting is to investigate the algorithm empirically. The tuning is performed using the *irace* package [27], which is an *R* framework based on the iterated racing procedure, an extension of the Iterated F-race procedure proposed by Birattari et al. [3].

The proposed simulated annealing has three influencing parameters: the number of trials *nb_trials*, the initial temperature T_0, and the cooling parameter α. According to multiple experiments, we observed that the number of trials *nb_trials* is highly dependent on the size of the instance. This is also due to the runtime limit set to 600 seconds as a standard among TTP algorithms. Therefore, this parameter must be chosen adaptively to the number of items m.

A first tuning attempt

In order to investigate the effect of the instance size on the three parameters and propose a first tuning technique, we consider eight instance groups of different sizes (Table 2). The idea consists of optimizing all the three parameters for each group of instances. The results of this experiment are reported in Table 3. According to the optimized values, the parameter *nb_trials* tends to change in an exponentially decreasing trend accordingly to the size of the instances. On the other hand, the values for α and T_0 vary widely in [0.90; 0.99] and [100; 1000] respectively. Even the elite configurations found by the irace package for the same group of instances vary unexpectedly in these ranges for the same group of instances. This leads us to believe that the size of the instance has a small influence on α and T_0.

Table 2
The eight groups of instances used for the tuning strategy, *bsc* stands for *bounded-strongly-correlated* KP type and *usw* stands for *uncorrelated-similar-weights* KP type.

Group name	Size range	Instances
SG1	1–129	eil51_n50_bsc_01, berlin52_n51_bsc_01, eil76_n75_bsc_01, kroA100_n99_bsc_01, ch130_n129_bsc_01
SG2	130–495	a280_n279_bsc_01 eil76_n375_usw_05, kroA100_n495_usw_05
SG3	496–990	u724_n723_bsc_01, dsj1000_n999_bsc_01, u159_n790_usw_05, eil76_n750_uncorr_10, kroA100_n990_uncorr_10
SG4	991–3037	rl1304_n1303_bsc_01, a280_n1395_usw_05, fl1577_n1576_bsc_01, u159_n1580_uncorr_10, d2103_n2102_bsc_01, a280_n2790_uncorr_10, u574_n2865_usw_05, pcb3038_n3037_bsc_01
SG5	3038–18,511	rl1304_n13030_uncorr_10, usa13509_n13508_bsc_01, brd14051_n14050_bsc_01, d15112_n15111_bsc_01, pcb3038_n15185_usw_05, fl1577_n15760_uncorr_10, d18512_n18511_bsc_01
SG6	18,512–75,555	rl11849_n59240_usw_05, usa13509_n67540_usw_05, brd14051_n70250_usw_05, pla7397_n73960_uncorr_10, d15112_n75555_usw_05
SG7	75,556–16,9045	usa13509_n135080_uncorr_10, brd14051_n140500_uncorr_10, d15112_n151110_uncorr_10, pla33810_n169045_usw_05
SG8	16,9046–$+\infty$	pla33810_n338090_uncorr_10

Table 3
A first tuning tentative for optimizing α, T_0, and *nb_trials* for each group of instances.

Group	α	T_0	*nb_trials*
SG1	0.9523	4052	$m \times 9566$
SG2	0.9148	4999	$m \times 9043$
SG3	0.9734	5791	$m \times 331$
SG4	0.9815	1479	$m \times 78$
SG5	0.984	124	$m \times 4$
SG6	0.9047	7138	$m \times 0.2876$
SG7	0.975	145	$m \times 0.0511$
SG8	0.9627	6272	$m \times 0.0285$

Table 4
Results for tuning *nb_trials* for each group of instances. α and T_0 are set to 0.95 and 300, respectively.

Group	*nb_trials*
SG1	$m \times 57872$
SG2	$m \times 13896$
SG3	$m \times 700$
SG4	$m \times 350$
SG5	$m \times 16$
SG6	$m \times 1$
SG7	$m \times 0.16$
SG8	$m \times 0.03$

Additionally, this approach presents drawbacks such as issues in covering all the instances efficiently. The preliminary results obtained for this configuration clearly confirm this statement. Furthermore, since *nb_trials* is highly dependent on the size of the instances, it would be more adequate to represent this parameter in terms of m.

A more advanced tuning strategy
In order to span all the instances more efficiently and avoid having a complex tuning scheme, we adopt the following steps:[5]

1. *nb_trials* is optimized for each one of the 8 groups of instances by fixing the values of α and T_0 to 0.95 and 300, respectively. The optimized values for the parameter *nb_trials* are shown in Table 4.
2. The outcoming values for *nb_trials* are then used in a spline interpolation formula in order to cover all the instance sizes more efficiently. The interpolation function, denoted L, is a concatenation of linear interpolants as expressed in Eq. (7).

[5] The reported values are the ones obtained for tuning CS2SA*. The same approach is used CS2SA-R, but different values were obtained.

Thus, the number of trials for a given number of items m is $nb\_trials = m \times L(m)$.

$$L(x) = \begin{cases} -341 \times x + 58,213 & \text{if } x \in [1, 130[\\ -36 \times x + 18,583 & \text{if } x \in [130, 496[\\ -0.71 \times x + 1050.71 & \text{if } x \in [496, 991[\\ -0.1631656082 \times x + 511.70 & \text{if } x \in [991, 3038[\\ -0.0009693680 \times x + 18.9449398992 & \text{if } x \in [3038, 18512[\\ -0.0000147255 \times x + 1.2725979945 & \text{if } x \in [18512, 75556[\\ -0.0000011841 \times x + 0.2494646401 & \text{if } x \in [75556, 169046[\\ -0.0000001142 \times x + 0.0686002283 & \text{if } x \in [169046, 338090[\\ 0.3 & \text{if } x \in [338090, +\infty[\end{cases} \quad (7)$$

Note that the interpolation points (knots) correspond to the size ranges in Table 2.

3. Finally, since α and T_0 are loosely dependent on m, an additional tuning round is performed to optimize these two parameters. The resulting values are 0.9578 and 98, respectively.

It is worth noting that although the *nb_trials* parameter changes in an exponential fashion according to the instance size, spline interpolation is shown to be significantly more stable than linear and exponential interpolation approaches.

Additionally, the same tuning procedure is used for *CS2SA-R*. Obviously, the resulting parameter values (step 1) are different from the ones obtained for *CS2SA**. Therefore, the interpolation function is also different from the one obtained for *CS2SA**. The most notable difference is that the number of trials are smaller. This is simply explained by the fact that *CS2SA-R* consists on repeating *CS2SA* as many times as possible, which leads to lower values for *nb_trials*.

4.3. Benchmark instances and experimental settings

The proposed algorithms are tested on instances of various sizes and properties from the TTP benchmark library proposed by Polyakovskiy et al. [36]. The TTP dataset introduces the following diversification parameters resulting in 9720 TTP instances:

- *TSP base problem:* All 81 TSPLIB instances [37].
- *Item factor (F):* 1, 3, 5, 10.
- *KP types (T):* uncorrelated (1), uncorrelated with similar weights (2), bounded strongly correlated (3).
- *Knapsack categories (C):* $1, \ldots, 10$.

Our tests are performed on a subset of the TTP library. In our setting, we consider the following TSP base problems: *eil76, kroA100, ch130, u159, a280, u574, u724, dsj1000, rl1304, fl1577, d2103, pcb3038, fnl4461, pla7397, rl11849, usa13509, brd14051, d15112, d18512, pla33810.*

The instances are divided into 3 categories:

- *Category 1 (F=1, T=3, C=1) :* 1 item per city, item values and weights are bounded and strongly correlated, small knapsack capacity.
- *Category 2 (F=5, T=2, C=5) :* 5 items per city, KP uncorrelated but items have similar weights, average knapsack capacity.
- *Category 3 (F=10, T=1, C=10) :* 10 items per city, KP uncorrelated, high knapsack capacity.

CS2SA variants[6] and *S5*[7] are mainly implemented in Java, while *MATLS* is implemented using C++.[8] 10 independent runs are performed for each instance. Note that all the algorithms have a maximum runtime limit of 600 seconds, which is used as a standard for TTP algorithms.

4.4. A comparative study

The results of the comparison are reported in Tables 5–7. The mean objective value (*mean*) is considered in order to compare the performance of algorithms in terms of solution quality. The relative standard deviation (*rsd*) is provided to measure the consistency, predictability and quality of the algorithms. The runtimes are mainly provided for guidance. Note that the runtimes (*time*) are measured in seconds, the *rsd* is given as a percentage, and the best mean objective values are highlighted in bold.

The results for the three categories show that *CS2SA**, *CS2SA-R*, and *S5* surpass *MATLS* for most TTP instances, especially for mid-size and very large instances (i.e. $m > 1000$). Another observation is that *CS2SA** performs better for large instances, while *CS2SA-R* is generally better for small instances.

Additionally, the runtimes for *CS2SA** is decent and predictable given the size for most instances. While in some executions, *MATLS* reaches the 600 s limit even for small instances.

[6] The source code for *CS2SA* implementations is publicly available at https://github.com/yafrani/ttplab

[7] For instance files and raw results for *S5* and other heuristics, the reader is referred to http://cs.adelaide.edu.au/~optlog/research/combinatorial.php

[8] Source code can be found at the website http://goanna.cs.rmit.edu.au/~e04499/Research-LSGO.html

Table 5
Results for category 1.

Instance	MATLS			S5			CS2SA*			CS2SA-R		
	Mean	rsd	Time	Mean	rsd	Time	Mean	rsd	Time	Mean	rsd	Time
eil76	3705	1.38	4.8	3742	0	600	3423	0.25	108	**3842**	2.33	600
kroA100	**4659**	1.34	67	4278	0	600	4420	1.07	143	4594	2.26	600
ch130	8876	0.83	62	**9250**	0	600	8982	0.42	161	9239	0.49	600
u159	8403	1.42	3.07	**8634**	0	600	8459	0	251	8560	0.97	600
a280	17,678	0.54	7.17	**18,406**	0.02	600	17,726	0.13	250	18,044	1.13	600
u574	26,121	2.15	193	**26,933**	0.14	600	26,173	1.06	113	25,383	2.4	600
u724	48,967	1.25	44	**50,316**	0.12	600	49,713	0.56	128	47,759	1	600
dsj1000	143,699	0	44	137,631	0.12	598	**144,219**	0	51	**144,219**	0	600
rl1304	75,800	1.26	63	**80,066**	0.94	600	75,699	0.47	205	73,557	1.38	600
fl1577	88,375	0.41	142	**92,343**	1.25	597	88,006	0.59	305	84,590	1.64	600
d2103	113,005	0.47	184	**120,642**	0.2	600	118,845	0.01	600	117,960	1.4	600
pcb3038	148,265	1.14	330	**160,006**	0.15	598	145,338	0.5	64	145,955	1.65	600
fnl4461	247,553	0.37	479	**262,237**	0.11	596	240,240	0.32	175	240,726	0.67	600
pla7397	365,506	1.11	578	**395,156**	0.56	591	315,154	0.01	600	325,852	9.95	600
rl11849	661,283	0.29	600	**707,190**	0.24	591	657,842	0.34	600	643,738	0.46	600
usa13509	747,905	0.47	600	**809,607**	0.24	582	682,268	0.11	600	700,243	3.7	600
brd14051	815,511	0.37	600	**875,018**	0.29	586	802,424	0.2	600	799,957	0.53	600
d15112	871,069	0.52	592	**939,790**	0.47	585	870,301	0.14	600	871,284	0.67	600
d18512	996,544	0.77	600	**1,072,308**	0.21	582	964,757	0.25	554	971,153	0.62	600
pla33810	1,730,207	0.87	600	**1,870,330**	0.78	572	1,778,827	0.24	600	1,668,597	1.83	600

Table 6
Results for category 2.

Instance	MATLS			S5			CS2SA*			CS2SA-R		
	Mean	rsd	Time	Mean	rsd	Time	Mean	rsd	Time	Mean	rsd	Time
eil76	**22,188**	0.78	3.2	20,097	0	600	18,753	0	231	22,032	1.2	600
kroA100	42,595	1.31	6.27	39,440	0	600	39,271	0	54	**43,712**	2.42	600
ch130	**61,061**	0.11	75	58,708	1.27	600	50,695	0	103	60,864	0.72	600
u159	58,289	1.33	10	57,618	0	600	58,090	0	123	**60,377**	1.03	600
a280	110,132	2.34	31	109,921	0.01	600	107,696	0	196	**114,087**	0.84	600
u574	**254,770**	0.71	222	251,775	0.05	600	248,584	0	116	248,402	1.05	600
u724	303,435	1.17	193	305,977	0.35	600	**309,636**	0	63	303,025	1.11	600
dsj1000	340,807	1.55	383	**342,189**	0.59	599	332,883	0	109	339,136	2.21	600
rl1304	572,766	1.18	290	575,102	0.86	600	**585,600**	0	181	578,064	1.41	600
fl1577	609,288	1.77	442	607,247	1.62	598	**636,425**	0	281	584,764	3.23	600
d2103	849,632	1.28	495	**853,587**	1.18	600	842,520	0	301	849,770	2.9	600
pcb3038	1,168,108	0.45	581	1,179,510	0.2	597	**1,193,738**	0	365	1,159,828	0.9	600
fnl4461	1,617,401	0.3	600	1,625,856	0.18	596	**1,628,414**	0	157	1,588,086	0.64	600
pla7397	4,178,551	3.87	600	**4,371,424**	0.79	591	3,713,314	0	383	3,798,791	9.37	600
rl11849	4,587,848	0.41	600	4,630,753	0.29	587	**4,710,135**	0	600	4,668,813	1.22	600
usa13509	7,767,305	2.02	600	7,818,124	0.72	579	**8,115,168**	0	600	7,930,181	2.54	600
brd14051	6,492,947	1.13	600	6,552,858	0.61	587	6,654,162	0	600	**6,667,470**	1.09	600
d15112	6,828,152	2	589	6,991,440	1.31	578	**7,606,856**	0	600	7,561,526	3.18	600
d18512	7,164,421	1.27	581	7,257,709	0.68	600	**7,579,996**	0.01	600	7,324,264	1.98	600
pla33810	15,532,990	1.1	600	15,574,552	0.73	566	**15,704,051**	0.5	600	14,421,765	4.51	600

4.5. Results analysis and discussion

According to the reported results, we believe that the proposed stochastic simulated annealing algorithm is well suited to solve the nonlinear knapsack component. It shows an excellent balance between exploitation and exploration. This is mainly due to the nature of the knapsack problem and the bit-flip operator. Indeed, modifying a picking plan requires changing the state of only one item. Thus, selecting the item randomly works well in contrast to TSP and the 2-OPT operator, which requires two cities to modify the tour – it is highly unlikely that both would be efficiently chosen randomly. Thus, following this naive reasoning, the 2-OPT neighborhood search is better suited for the TSP component. We ran some tests to confirm this by comparing a heuristic selecting two cities randomly against the 2-OPT neighborhood search. The results confirm that the 2-OPT search offers a better quality/runtime ratio than the random selection.

However, the simple 2-OPT heuristic proposed in our implementation still suffers from a lack of exploration. Although the KP component has the most significant impact on the overall problem as previously shown, a more sophisticated heuristic for the TSP component would be more preferred. For instance, in *MATLS* the order crossover operator [19] is used to tackle this problem and explore more possibilities. While Wagner [40] proposed to use a ants to generate tours and exploit the fact that ant systems are not very efficient in solving the TSP. Stating with mediocre tours allows to avoid the bias, and

Table 7
Results for category 3.

Instance	MATLS			S5			CS2SA*			CS2SA-R		
	Mean	rsd	Time	Mean	rsd	Time	Mean	rsd	Time	Mean	rsd	Time
eil76	**88,136**	0.26	18	85,664	0	600	87,577	0	49	88,099	0.36	600
kroA100	155,492	0.02	203	155,540	0	600	155,585	0	77	**155,878**	0.53	600
ch130	203,468	2.21	32	201,174	0.85	600	197,555	0	75	**206,561**	0.15	600
u159	242,567	0.43	23	242,495	0.31	600	242,201	0	103	**249,911**	0.06	600
a280	426,259	0.2	34	**429,000**	0	600	421,713	0	78	427,056	0.14	600
u574	966,207	0.3	134	**966,344**	0.05	600	953,997	0	68	952,984	0.4	600
u724	1,188,761	0.4	364	1,188,364	0.14	600	**1,191,819**	0	71	1,181,819	0.36	600
dsj1000	1,472,638	1	516	**1,479,618**	0.1	600	1,468,858	0	152	1,464,474	0.74	600
rl1304	2,178,475	0.21	476	2,184,853	0.21	600	**2,198,943**	0	188	2,165,629	0.68	600
fl1577	2,466,403	0.42	600	2,470,917	0.36	598	**2,505,291**	0	150	2,382,050	1.88	600
d2103	**3,392,877**	0.25	600	3,392,172	0.15	599	3,373,781	0	75	3,331,873	0.97	600
pcb3038	4,564,261	0.19	600	4,573,748	0.09	596	**4,612,956**	0	148	4,514,051	0.32	600
fnl4461	6,534,422	0.23	600	**6,554,497**	0.1	592	6,545,335	0	285	6,463,969	0.22	600
pla7397	13,865,880	2.16	600	**14,239,609**	1.01	569	13,197,756	0	242	13,808,867	3.94	600
rl11849	18,275,210	0.17	597	18,314,650	0.16	582	**18,504,773**	0	600	18,203,774	0.49	600
usa13509	25,878,290	0.53	595	25,918,971	0.4	568	**26,436,928**	0	595	26,251,061	0.67	600
brd14051	23,672,310	0.51	600	23,826,398	0.5	577	**23,908,555**	0	600	23,797,888	0.95	600
d15112	25,942,410	1.11	590	26,211,266	0.87	577	**27,183,354**	0	600	27,160,800	1.32	600
d18512	27,164,500	1.07	596	27,427,135	0.35	569	**27,823,470**	0.11	600	27,269,855	1.06	600
pla33810	58,003,980	0.4	600	57,967,446	0.39	541	**58,106,820**	0.01	600	55,631,820	1.38	600

the resulting algorithm was very competitive for small instances. To address this issue, multiple other solutions could be considered. The hybridization with an evolutionary algorithm is from our point of view the most obvious. In fact, we believe that our approach shows promising hybridization opportunities only by performing little changes in our heuristics in order to reduce the runtime (e.g. reduce the number of trials in SA). Additionally, other metaheuristics could be considered such as tabu search [17,18], the 3-OPT heuristic [25], the ant colony optimization algorithm [10], and the threshold accepting algorithm [12] to name a few.

As explained in Section 3, the heuristics used for the TTP are extremely sensitive to the initial solution, especially the tour. Thus, using an evolutionary algorithm with a population of N solutions would require N good initial tours generated with some time-consuming heuristic such as the Chained Lin–Kernighan. Moreover, the initial tours should be mutually nonidentical, which implies using a stochastic heuristic and implementing a mechanism to eliminate redundant tours. In *MATLS*, the Lin–Kernighan heuristic and a Spanning Tree algorithm are used to generate the tours for an initial population of 30 solutions. We observed that this process takes a considerable amount of time, especially for very large instances. In fact, the initialization time is so significant for large instances that it does not leave enough time for evolving the population, leading to a very low computational effort during the evolution. One advantage of using a single solution heuristic over an evolutionary algorithm for the TTP is that only one initial tour is required. Thus, the initialization heuristic is used only once, which saves a significant amount of time that can be better utilized in the search process. This is one of the main features making *CS2SA**, *CS2SA-R*, and *S5* outperform *MATLS*.

The reported results also show that *S5* surpasses the other algorithms for most instances from the first category. A possible explanation could be related to the technique used by *S5* to tackle the KP sub-problem. *S5* uses a greedy routine based on sorting items according to their profit, weight, and remoteness from the last city. Given that this category has a small number of items (1 item per city) and the smallest knapsack capacity, we believe such a greedy approach is beneficial in solving this particular type of KP, even when the weights and values of the items are highly correlated. Additional experiments were performed on instances with a low correlation rate and the obtained results show that *S5* performed similarly well for these instances. This aspect has also been explored empirically in [14], and it has been suggested that the most influencing KP parameter is the knapsack capacity C.

For the instances with a higher knapsack capacity (categories 2 and 3), *S5* is still competitive among the other heuristics. However, *CS2SA** has a better performance for most instances. Tables 6 and 7 show that *CS2SA** clearly outperforms the other heuristics for the majority of instances, while *CS2SA-R* is competitive for some small-size instances, which is not surprising as it was designed to exploit the entire time budget using a different tuning. As these instances have a high knapsack capacity, we believe that the superior performance of *CS2SA** is due to its stochastic behaviour based on selecting items randomly, while using the Boltzmann criterion to manage the balance between exploration (early stage of the search) and exploitation of particular search areas. This process works particularly well for high knapsack capacities as this allows us to explore more possibilities for combining items.

In order to gain further insights into the performance of the four algorithms, we use the *Vargha–Delaney A-test* [39]. The *A-test* returns a value between 0 and 1, called the *A-measure*, representing the probability that a randomly selected observation from one sample is better than a randomly selected observation from the other sample. When the *A-measure*

Table 8
A summary table representing the success rate of each algorithm over the others for all TTP instances from the three categories, given the obtained A-measures.

Algorithm 1	Algorithm 2	Category 1	Category 2	Category 3
CS2SA*	**MATLS**	0.5	**0.6**	**0.6**
CS2SA*	**S5**	0.1	**0.6**	**0.6**
CS2SA-R	**MATLS**	0.3	0.5	0.5
CS2SA-R	**S5**	0.15	**0.55**	0.3
CS2SA*	**CS2SA-R**	0.45	**0.55**	**0.65**
MATLS	**S5**	0.1	0.3	0.3

is exactly 0.5, there is no statistical difference between the algorithms. When the *A-measure* is lower than 0.5, the first algorithm has the worst performance. Otherwise, the second algorithm is the worst performing one.

In our case, the samples are consisting of 10 objective values, corresponding to 10 independent runs. Table 8 represents a summary of a pairwise comparison using multiple *A-measures* for all the instances from the three categories. Each cell represents a success rate of the first algorithm over the second. The success rate is calculated by counting the number of times the *A-measure* is higher than 0.5 divided by 20, the total number of instances in one category. The cells corresponding to a success rate strictly greater than 0.5 are emphasized.

The results reported in Table 8 confirm clearly that *S5* has the best performance on average for the instances from the first category, while *CS2SA** outperforms all the other algorithms on average for the second and third categories.

The strength of *S5* relies on the use of a greedy routine based on efficient item sorting. On the other hand, the potential of our two heuristics and *MATLS* is mainly due to making strong assumptions about the problem domain, mainly concretized using fast solution evaluation. This strategy limits their applicability to other problems. However, we believe that the transition to other multi-component routing problems [24,34,38] is facilitated with these works on the TTP. Also, as explained by Mei et al. [30], solving the TTP would directly help solving the Capacitated Arc Routing Problem [11].

5. Concluding remarks

In this paper, we proposed two algorithms that combine the 2-OPT steepest ascent hill climbing heuristic and a simulated annealing algorithm named *CS2SA** and *CS2SA-R*. Our approaches are based on the CoSolver framework and multiple performance enhancement techniques. We carried out an empirical study by comparing our algorithms against two state-of-the-art heuristics, namely *MATLS* and *S5*. The obtained results show that our two heuristics were competitive and able to surpass *MATLS* and *S5* for many instances of various sizes and types. While *CS2SA** was competitive for most instances, it still presents drawbacks for instances with a low knapsack capacity.

In the future, further efforts will be made to investigate other problems with multiple interdependent components in order to gain more insights about the internal dependencies in real-world problems. The TTP has been criticized for not being more realistic. In [7], the authors proposed a relaxed version of the problem where multiple thieves are allowed as an attempt to make the TTP more realistic. However, we believe that the fact the TTP only has two components is another limitation. In fact, most real-world supply-chain optimization problems have many components. Therefore, we engage to investigate problems with many interacting component that have potential real-world applications.

References

[1] E. Aarts, J. Korst, Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing, John Wiley & Sons, Inc., New York, NY, USA, 1989.
[2] D. Applegate, W. Cook, A. Rohe, Chained Lin–Kernighan for large traveling salesman problems, INFORMS J. Comput. 15 (1) (2003) 82–92.
[3] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: an overview, in: Experimental Methods for the Analysis of Optimization Algorithms, Springer, 2010, pp. 311–336.
[4] M.R. Bonyadi, Z. Michalewicz, L. Barone, The travelling thief problem: the first step in the transition from theoretical problems to realistic problems, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2013, pp. 1037–1044.
[5] M.R. Bonyadi, Z. Michalewicz, F. Neumann, M. Wagner, Evolutionary computation for multi-component problems: opportunities and future directions, CoRR, 2016, http://arxiv.org/abs/1606.06818.
[6] M.R. Bonyadi, Z. Michalewicz, M. Roman Przybyõek, A. Wierzbicki, Socially inspired algorithms for the Travelling Thief Problem, in: Proceedings of the Conference on Genetic and Evolutionary Computation, ACM, 2014, pp. 421–428.
[7] S. Chand, M. Wagner, Fast heuristics for the multiple traveling thieves problem, in: Proceedings of the Conference on Genetic and Evolutionary Computation, ACM, 2016, pp. 293–300.
[8] G. Croes, A method for solving traveling-salesman problems, Oper. Res. 6 (6) (1958) 791–812.
[9] B. Delaunay, Sur la sphere vide, Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7 (793–800) (1934) 1–2.
[10] M. Dorigo, L.M. Gambardella, Ant colonies for the travelling salesman problem, BioSystems 43 (2) (1997) 73–81.
[11] M. Dror, Arc Routing: Theory, Solutions and Applications, Springer Science & Business Media, 2012.
[12] G. Dueck, T. Scheuer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, J. Comput. Phys. 90 (1) (1990) 161–175.
[13] M. El Yafrani, B. Ahiod, Cosolver2b: an efficient local search heuristic for the travelling thief problem, in: Proceedings of the IEEE/ACS Twelfth International Conference on Computer Systems and Applications (AICCSA), IEEE, 2015, pp. 1–5.
[14] M. El Yafrani, B. Ahiod, Population-based vs. single-solution heuristics for the Travelling Thief Problem, in: Proceedings of the Conference on Genetic and Evolutionary Computation, in: GECCO'16, ACM, Denver, CO, USA, 2016.

[15] M. El Yafrani, B. Ahiod, A local search based approach for solving the Travelling Thief Problem: the pros and cons, Appl. Soft Comput. 52 (2017) 795–804.
[16] H. Faulkner, S. Polyakovskiy, T. Schultz, M. Wagner, Approximate approaches to the Traveling Thief Problem, in: Proceedings of the Conference on Genetic and Evolutionary Computation, in: GECCO'16, ACM, Denver, CO, USA, 2016.
[17] F. Glover, Tabu search-part i, ORSA J. Comput. 1 (3) (1989) 190–206.
[18] F. Glover, Tabu search-part ii, ORSA J. Comput. 2 (1) (1990) 4–32.
[19] D.E. Goldberg, R. Lingle, Alleles, loci, and the traveling salesman problem, in: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Publishers, 1985, pp. 154–159.
[20] L. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi, ACM Trans. Gr. (TOG) 4 (2) (1985) 74–123.
[21] J.-Q. Hu, B. Leida, Traffic grooming, routing, and wavelength assignment in optical WDM mesh networks, in: Proceedings of the INFOCOM Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies, 1, IEEE, 2004.
[22] M. Ibrahimov, A. Mohais, S. Schellenberg, Z. Michalewicz, Evolutionary approaches for supply chain optimisation: part I: single and two-component supply chains, Int. J. Intell. Comput. Cybern. 5 (4) (2012) 444–472.
[23] M. Ibrahimov, A. Mohais, S. Schellenberg, Z. Michalewicz, Evolutionary approaches for supply chain optimisation. part II: multi-silo supply chains, Int. J. Intell. Comput. Cybern. 5 (4) (2012) 473–499.
[24] M. Iori, S. Martello, Routing problems with loading constraints, Top 18 (1) (2010) 4–27.
[25] S. Lin, Computer solutions of the traveling salesman problem, Bell Syst. Tech. J. 44 (10) (1965) 2245–2269.
[26] S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Oper. Res. 21 (2) (1973) 498–516.
[27] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration, Technical Report, Citeseer, 2011.
[28] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, Inc., 1990.
[29] Y. Mei, X. Li, X. Yao, Improving efficiency of heuristics for the large scale Traveling Thief Problem, in: Simulated Evolution and Learning, Springer, 2014, pp. 631–643.
[30] Y. Mei, X. Li, X. Yao, On investigation of interdependence between sub-problems of the travelling thief problem, Soft Comput. (2014) 1–16.
[31] Z. Michalewicz, Genetic Algorithms+ Data Structures= Evolution Programs, Springer Science & Business Media, 2013.
[32] F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, J. Wu, A fully polynomial time approximation scheme for packing while traveling, CoRR, abs/1606.06818 (2017). http://arxiv.org/abs/1702.05217.
[33] C.H. Papadimitriou, The euclidean travelling salesman problem is np-complete, Theor. Comput. Sci. 4 (3) (1977) 237–244.
[34] S. Polyakovskiy, F. Neumann, Packing while traveling: mixed integer programming for a class of nonlinear knapsack problems, in: Integration of AI and OR Techniques in Constraint Programming, Springer, 2015, pp. 332–346.
[35] S. Polyakovskiy, F. Neumann, The packing while traveling problem, Eur. J. Oper. Res. 258 (2) (2017) 424–439.
[36] S. Polyakovskiy, M. Reza, M. Wagner, Z. Michalewicz, F. Neumann, A comprehensive benchmark set and heuristics for the Traveling Thief Problem, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Vancouver, Canada, 2014.
[37] G. Reinelt, TSPLIB – A traveling salesman problem library, ORSA J. Comput. 3 (4) (1991) 376–384.
[38] J. Stolk, I. Mann, A. Mohais, Z. Michalewicz, Combining vehicle routing and packing for optimal delivery schedules of water tanks, OR Insight 26 (3) (2013) 167–190.
[39] A. Vargha, H.D. Delaney, A critique and improvement of the CL common language effect size statistics of Mcgraw and Wong, J. Edu. Behav. Stat. 25 (2) (2000) 101–132.
[40] M. Wagner, Stealing items more efficiently with ants: a swarm intelligence approach to the Travelling Thief Problem, in: Proceedings of the tenth International Conference on Swarm Intelligence, ANTS 2016, Springer International Publishing, Brussels, Belgium, 2016, pp. 273–281.
[41] M. Wagner, M. Lindauer, M. Misir, S. Nallaperuma, F. Hutter, A case study of algorithm selection for the Traveling Thief Problem, J. Heuristics (2017).
[42] M. Weiszer, J. Chen, S. Ravizza, J. Atkin, P. Stewart, A heuristic approach to greener airport ground movement, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 3280–3286.
[43] J. Wu, S. Polyakovskiy, F. Neumann, On the impact of the renting rate for the unconstrained nonlinear knapsack problem, in: Proceedings of the Conference on Genetic and Evolutionary Computation, in: GECCO'16, ACM, Denver, CO, USA, 2016.