A Comprehensive Benchmark Set and Heuristics for the Traveling Thief Problem

Sergey Polyakovskiy
School of Computer Science
The University of Adelaide

Mohammad Reza Bonyadi
School of Computer Science
The University of Adelaide

Markus Wagner
School of Computer Science
The University of Adelaide

Zbigniew Michalewicz
School of Computer Science
The University of Adelaide

Frank Neumann
School of Computer Science
The University of Adelaide

ABSTRACT

Real-world optimization problems often consist of several NP-hard optimization problems that interact with each other. The goal of this paper is to provide a benchmark suite that promotes a research of the interaction between problems and their mutual influence. We establish a comprehensive benchmark suite for the traveling thief problem (TTP) which combines the traveling salesman problem and the knapsack problem. Our benchmark suite builds on common benchmarks for the two sub-problems which grant a basis to examine the potential hardness imposed by combining the two classical problems. Furthermore, we present some simple heuristics for TTP and their results on our benchmark suite.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Miscellaneous; I.2.8 [**Problem Solving, Control Methods, and Search**]: Scheduling

General Terms

Experimentation, Algorithms

Keywords

Traveling thief problem, knapsack problem, interdependence, benchmarks

1. INTRODUCTION

Real-world optimization problems usually consist of several problems that interact with each other. In order to solve such problems it is important to understand and deal with these interactions. So far, the research literature is lacking systematic approaches for dealing with such interdependent problems.

In this paper, we consider the Traveling Thief Problem (TTP) [4] which is a combination of two of the most prominent combinatorial optimization problems, namely the traveling salesperson problem (TSP) and the knapsack problem (KP). Both problems have been considered in numerous theoretical and experimental studies, and very effective solvers are known that perform well on a variety of benchmarks.

We present a benchmark suite for TTP which is based on benchmark instances for the TSP and KP. The aim of this benchmark suite is to give researchers the opportunities

- to study TTP as a combination of TSP and KP,
- to compare the problem instances, and
- to provide algorithms that can effectively solve problems with interdependencies.

We systematically construct the benchmarks so that the resulting instances cover a wide range of features: from few cities with few items and small knapsacks to many cities with many items and large knapsacks. While we are confident that smaller instances can soon be solved to optimality, the larger ones most likely remain unsolved for the years to come.

It is important to note that the TTP is unlike many capacitated vehicle-routing problems in the area of Green Logistics (see, e.g., the survey article [8]). For example, the fuel consumption based on load or geographical features is considered in [6, 7] and the problem for several vehicles is solved using integer programming. In our case, we add to the routing problem not only a load-dependent feature, but also the NP-hard optimization problem of deciding which items are to be packed.

The remainder of this paper is organized as follows. First, we define the traveling thief problem in Section 2. In Section 3 we describe the systematic construction of the benchmark set. We present our heuristic algorithms in Section 4 and their results on the benchmarks in Section 5. We finish with some concluding remarks.

2. TRAVELING THIEF PROBLEM

The Traveling Thief Problem is defined as follows. Given is a set of cities $N = \{1, \dots, n\}$ for which a distance d_{ij}, $i, j \in N$ between any pair of cities is known. Every city i but the first contains a set of items $M_i = \{1, \dots, m_i\}$. Each item k positioned in the city i is characterized by its value p_{ik} and weight w_{ik}, thus the item $I_{ik} \sim (p_{ik}, w_{ik})$. The thief must visit each of the cities exactly once starting from the first

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598249.

city and returning back to it in the end. Any item may be picked up into the knapsack in any city until the total weight of collected items does not exceed the maximum possible weight W. A renting rate R is to be paid per each time unit being on a way. υ_{max} and υ_{min} denote the maximal and minimum speeds that the thief can move, respectively. The goal is to find a tour of the maximal profit.

Let $y_{ik} \in \{0, 1\}$ be a binary variable equal to one when the item k is picked up in the city i. In addition, let W_i denote the total weight of collected items when the thief leaves the city i. Therefore, the objective function for a tour $\Pi = (x_1, \ldots, x_n)$, $x_i \in N$ and a packing plan $P = (y_{21}, \ldots, y_{nm_i})$ takes the following form:

$$\begin{aligned} Z(\Pi, P) &= \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} \\ &\quad -R \left(\frac{d_{x_n x_1}}{\upsilon_{max} - \nu W_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\upsilon_{max} - \nu W_{x_i}} \right) \end{aligned}$$

where $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$ is a constant value. The minuend is the sum over all packed items' profits and the subtrahend is the amount that the thief pays for the knapsack's rent equal to the total traveling time along Π multiplied by R.

A numeric example of the TTP problem is provided via the graph given in Figure 1. Each node but the first has an assigned set of items, e.g. node 2 is associated with item I_{21} of profit $p_{21} = 20$ and weight $w_{21} = 2$, and with item I_{22} of profit $p_{22} = 30$ and weight $w_{22} = 3$. Let's assume that the maximum weight $W = 3$, the renting rate $R = 1$ and υ_{max} and υ_{min} are set as 1 and 0.1, respectively. Then the optimum objective value $Z(\Pi, P) = 50$ for $\Pi = (1, 2, 4, 3)$ and $P = (0, 0, 0, 1, 1, 0)$. Specifically, the thief collects no items traveling from city 1 to city 3 via cities 2 and 4. Therefore, this part of the tour costs 15. Only in the city 3 items I_{32} and I_{33} are picked up, that gives total profit of 80. However, on the way from city 3 back to city 1 the thief's knapsack has weight of 2. In fact, it reduces the speed and results in increased cost of 15. Consequently, the final objective value is $Z(\Pi, P) = 80 - 15 - 15 = 50$.

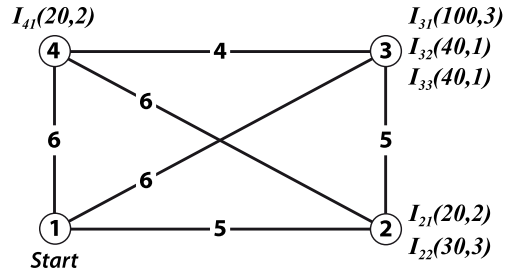

Figure 1: Illustrative Example

3. BENCHMARK SET

Herein we propose a new benchmark of instances in order to promote a correct comparison among different approaches to the TTP problem. It is mainly motivated by a desire to (1) have a combination of well-known test suites for each sub-problem, and (2) a correct adjustment of constant variables involved in the objective function. Specifically, from an experimental point of view, it is important to keep a balance between two components of the problem. It means that the near-optimal solution of one sub-problem must not dominate over the optimal solution of another sub-problem. Therefore, solving the shortest tour problem to optimality must not make the knapsack packing aspect negligible. Vice versa, the most profitable loading plan must not reduce the importance of a shorter tour.

To handle this task the most known set of test instances for each sub-problem was considered. The TSP library collected by Reinelt [2, 10] is to be the main departure point when one is going to evaluate his/her approach's performance on the TSP. It consists of over 100 instances with sizes ranging from 14 to 85,900 cities and is derived from industrial applications and geographic problems. Since its creation in 1990, an extensive investigation has been carried out and now all the symmetric TSP library instances are solved to optimality.

The knapsack problem was thoroughly studied by Martello, Pisinger and Toth [9]. Their work deals with the 0-1 knapsack and proposes the exact dynamic programming algorithm called `COMBO` to tackle it. Furthermore, they propose and investigate a test benchmark wealthy in ways to generate KP data. `COMBO` adopts a set of valid inequalities along with a new initial core problem that allows it to solve instances with up to 10,000 items in less than 0.2 seconds. The authors generated fifteen different types of KP varying weights and profits of items to investigate which aspects result in challenges for KP solvers in general and for their approach in particular. From their work, we pick several of the hardest cases and combine them with instances of the TSP library.

To generate our TTP instances we first consider the instances of the TSP library. First, we select the 81 instances that possess "EUC_2D" or "CEIL_2D" marks for their edge weight type tag. Despite the mark derived from the TSP library we assume that over the entire TTP benchmark suite all distances between the cities are integer values rounded up to the next integer. The resulting set has instances with the number of cities ranging from 51 and up to 85,900.

For each TSP instance, we generate a knapsack component using the code from [1]. Three ways to generate a KP data are selected respecting the evaluated hardness for solving and the recommendations reported in [9]. Each way results in a unique knapsack type. In the following, we distinguish between *uncorrelated*, *uncorrelated with similar weights* and *bounded strongly correlated* types.

For the uncorrelated type a weight w_{ik} and a profit p_{ik} of item k are uniformly distributed random integer values in $\left[1, 10^3\right]$. Loose correlation or no correlation among the profit and weight of each item makes an instance easy to solve to optimality even for large-sized problems [9]. However, this may be not straightforward for the TTP problem and a behavior of algorithms stays of a particular interest for research.

Addressing the uncorrelated type with similar weights we employ for w_{ik} and p_{ik} integer values uniformly distributed within $\left[10^3, 10^3 + 10\right]$ and $\left[1, 10^3\right]$ respectively. Producing items with similar weights complicates KP by increasing the core problem's size. It has already been shown that this type of 0-1 knapsack instances is more time consuming to be solved [9]. Consequently, having all items of almost equal weight but definitely different profit may require an addi-

tional effort to develop an algorithm which intelligently deal with this aspect in TTP, too.

Oppositely to the uncorrelated case, strongly correlated problems are typically very difficult. The bounded strongly correlated type adopts the bounded knapsack problem where multiple copies of an item are allowed to be selected complying with a maximum number restriction. As previously shown, its transformation to a 0-1 knapsack problem complicates the problem [9]. It significantly increases the computation time while it forces the cardinality constraints in COMBO to loose their effect. This indicates that specialized algorithms for the bounded version should be developed. To set up the initial bounded instance, we choose the interval $\left[1, 10^3\right]$ for an item's integer weight w_{ik} uniformly distributed generation and we set p_{ik} based on this as $w_{ik}+100$.

To diversify the size of the knapsack component, we distinguish between four values of an *item factor* $F \in \{1, 3, 5, 10\}$. In our benchmarks, F describes how many items per city are available. We use the same number of items for every city setting $F = m_i$ for any $i \in N$, $i \geq 2$. As assumed in the problem statement, the first city does not contain any item. Therefore, for a number of n cities we produce totally $F \times (n-1)$ items. Moreover, no correlation exists between the item's features and the features of the city it is assigned to, i.e. a simple many-to-one assignment between the set of cities and the set of items is performed, where each item $(i + f \times (n-1))$, $f \in \mathbb{N}^0$, $i \in \{1, \ldots, n-1\}$, goes to the city $(i+1)$.

Next, for each TSP library instance, knapsack type and item factor we generate 10 instances. Each instance falls into its *capacity category* $C \in \{1, \ldots, 10\}$ according to the instance number. The capacity category value C results in a unique maximum Knapsack weight value of $W = \left(\frac{C}{11}\right) \sum_{i=1}^{n} \sum_{k=1}^{F} w_{ik}$. Effectively, C is the factor by which the capacity of the smallest knapsack (in the set of 10) is multiplied.

As noted above, we try to manage the potential objective value by searching for a balance between TSP and KP components through the constant variables. Specifically, we set the same values for $\upsilon_{max} = 1$ and $\upsilon_{min} = 0.1$ over all instances. However, each instance gets its individual renting rate R defined as $R = \frac{Z\left(P^{OPT}\right)}{TIME\left(\Pi^{linkern}, P^{OPT}\right)}$, where $Z\left(P^{OPT}\right)$ corresponds to the optimal profit of the KP component and $TIME\left(\Pi^{linkern}, P^{OPT}\right)$ denotes the total traveling time along the near-optimal TSP tour $\Pi^{linkern}$ obtained via the Chained Lin-Kernighan heuristic [3][1] while picking the items according to the optimal KP component's solution P^{OPT}. Such selection of R guarantees the existence of at least one TTP solution with zero objective value.

Consequently, our final benchmark set based on 81 TSP instances, three KP types, four item factors and ten capacity categories contains $81 \times 3 \times 4 \times 10 = 9,720$ different TTP instances in total.[2]

4. HEURISTIC ALGORITHMS

[1]As available at http://www.tsp.gatech.edu/concorde/downloads/downloads.htm

[2]All instances and implementations of the objective function are available online: http://cs.adelaide.edu.au/~optlog/research/ttp.php

Algorithm 1 Simple Heuristic (SH)

```
Fill the array D with values d_{x_i}, x_i ∈ {x_2, ..., x_n}
Calculate the total traveling time t'.
for all items I_{x_ik}, x_i ∈ Π, k ∈ M_{x_i} do
    Calculate t_{x_ik} by using Equation 1
    Set t'_{x_ik} := t' − d_{x_i} + t_{x_ik}
    Set score_{x_ik} := p_{x_ik} − R × t_{x_ik}
    Set u_{x_ik} := R × t' + (p_{x_ik} − R × t'_{x_ik})
Create the joint set of items I and sort them in descending order score values
Set the current used capacity variable Wc := 0
for all items I_{x_ik} ∈ I do
    if (Wc + w_{x_ik} < W) and (u_{x_ik} > 0) then
        Add the item I_{x_ik} to the packing plan P
        Increase the used capacity variable Wc := Wc + w_{x_ik}
    if (Wc = W) then
        Exit the loop.
Set the resulting objective value
Z* := max(Z(Π, P), −R × t')
```

We create solutions for TTP instances in two stages. First, we use the Chained Lin-Kernighan heuristic [3] to quickly construct a good TSP tour. We do this without considering the knapsack part of the TTP problem. Once we have this tour, it remains fixed for the duration of the subsequent optimization. Second, a heuristic creates a packing plan in order to achieve a good TTP objective value. The rest of the section presents our constructive and iterative heuristics.

4.1 Constructive Heuristic

This section describes a Simple Heuristic (SH) which constructs a solution by processing and picking the items that maximize the objective value according to a given tour. In SH, after generating a tour $\Pi = (x_1, \ldots, x_n)$ for the TSP problem, a *score* value is calculated for each item to estimate how good it is according to Π. Let item I_{x_ik} be positioned in city $x_i \in \Pi$. Subsequently, let d_{x_i} and t_{x_ik} denote the total traveling distance and the total traveling time with I_{x_ik} being collected from the city x_i to x_n and from x_n to the first city x_1, respectively. Then the function to calculate $score_{x_ik}$ is as follows:

$$score_{x_ik} = p_{x_ik} - R \times t_{x_ik}$$

where

$$t_{x_ik} = \frac{d_{x_i}}{\upsilon_{max} - \nu w_{x_ik}} \quad (1)$$

Specifically, $score_{x_ik}$ is the total profit that the thief gets if and only if item I_{x_ik} is picked during the whole tour Π.

In SH we also use a strategy to eliminate so-called *disadvantageous* items which being taken cannot add any positive value to the resulting objective value. This strategy utilizes a fitness value u_{x_ik} calculated for each item I_{x_ik} as follows:

$$u_{x_ik} = R \times t' + (p_{x_ik} - R \times t'_{x_ik})$$

where t'_{x_ik} and $t' = \sum_{i=1}^{n-1} d_{x_i x_{i+1}} + d_{x_n x_1}$ are the total traveling times over the tour Π when only item I_{x_ik} is picked and when no items at all are picked (i.e. with maximal

Algorithm 2 Random Local Search (RLS)

1: Initialize P^* such that no items are packed.
2: **repeat until no improvement for** X **iterations**
3: Create P by inverting the packing status of a uniformly at random picked item of P^*.
4: **if** $Z(\Pi, P) \geq Z(\Pi, P^*)$ and $w(P) \leq W$ **then**
5: $P^* := P$

Algorithm 3 (1+1) Evolutionary Algorithm (EA)

1: Initialize P^* such that no items are packed.
2: **repeat until no improvement for** X **iterations**
3: Create P by inverting the packing status of each item of P^* independently with probability $1/m$.
4: **if** $Z(\Pi, P) \geq Z(\Pi, P^*)$ and $w(P) \leq W$ **then**
5: $P^* := P$

possible speed υ_{max}). Therefore, a negative value of u_{x_ik} means that I_{x_ik} must not be picked up despite any obtained score value. For a more detailed explanation, let us assume in the following that $u_{x_ik} \leq 0$ is the case. Let $\Delta\left(t'_{x_ik}, t'\right) = t'_{x_ik} - t'$ show the difference between the traveling times when I_{x_ik} and no other item is selected. Then the inequality $p_{x_ik} \leq R \times \Delta\left(t'_{x_ik}, t'\right)$ holds. Subsequently, let I^* denote an arbitrary non-empty set of items that fit into the knapsack, while t'_{I^*} and $t'_{I^* \cup I_{x_ik}}$ be the total traveling times over the tour Π when only items of I^* are picked and when items of I^* along with item I_{x_ik} are chosen, respectively. As the velocity depends linearly on the weight of the knapsack, the inequality $\Delta\left(t'_{x_ik}, t'\right) < \Delta\left(t'_{I^* \cup I_{x_ik}}, t'_{I^*}\right)$ is satisfied. Therefore, $p_{x_ik} < R \times \Delta\left(t'_{I^* \cup I_{x_ik}}, t'_{I^*}\right)$ proves that the preposition holds for any non-empty set I^* picked along with the disadvantageous item.

Once the computations of the u_{x_ik} and $score_{x_ik}$ values for every item are finished, the joint set of all items I is sorted in descending order of the items' scores. Subsequently, each item of the joint set is considered one after another starting from the first element of I. If item I_{x_ik} fits into the free capacity of the knapsack while its fitness value $u_{x_ik} > 0$ then it is taken to the packing plan P. Finally, SH obtains the resulting objective value as $Z^* = max\left(Z(\Pi, P), -R \times t'\right)$ that corresponds to the best choice between constructed solution and solution with no selected items (see Algorithm 1).

Let $m = |I|$ denote the total number of items of a TTP instance. Then the computational complexity of SH for constructing a packing plan P for a pre-set tour Π is in $O(n + m \log m)$. Specifically, it takes $O(n)$ to compute elements of array $D = \{d_{x_2}, \ldots, d_{x_n}\}$. Calculation of score and fitness values is in $O(m)$, while sorting of items and selecting items are in $O(m \log m)$ and $O(m)$, respectively.

4.2 Iterative Heuristics

Our iterative heuristics are a random local search (RLS) and a simple (1+1) evolutionary algorithm (EA). In stark contrast to the previously proposed constructive SH, the iterative heuristics do not use any domain knowledge whatsoever.

The heuristics work as follows. They record the best solution found so far, known as best-so-far. They then repeatedly create a single new solution. This solution is then compared against the current best-so-far and it replaces the previous best-so-far if the TTP objective value is not worse and if the knapsack capacity is not violated.

Formal descriptions are given in Algorithm 2 and 3, where P is a packing plan, m is the total number of items, $Z(\Pi, P)$ is the TTP objective value (assuming the given and fixed tour Π), and $w(P)$ is the total weight of all packed items.

5. EXPERIMENTS

The experimental setup is the follows. We run our heuristic algorithms from Section 4 on all 9,720 instances. The iterative heuristics are stopped when no improvement has been made for 10,000 iterations, or when a total runtime of 10 minutes for a run is reached. Due to their randomized nature, we perform 30 independent repetitions of the iterative heuristics on each instance.

First, to begin with our analyses, we show in Figure 2 a decision tree that is based on all gathered experimental data. It is generated with Matlab's Statistics Toolbox and we pruned the tree in order to get a compact tree that still contains valuable information to a practitioner. We observe the following:

- At first sight, it appears that the number of cities plays no role in this particular decision tree. However, this information is implicitly used via the number of items and the knapsack capacity category.
- In general, the evolutionary algorithm EA dominates small TTP instances with few cities and few items, as many of the leftmost leaves in the decision tree are labelled EA. It performs better than RLS since it can escape local optima.
- The random local search RLS performs well across mid-sized and several larger TTP instances. This is due to the fact that EA effectively wastes evaluations, as the packing status change of an item is not enforced. Consequently, RLS is more effective in reaching a local optima, as a change in the newly generated packing plan is enforced.
- The deterministic constructive heuristic SH can be found at the right-most leaf nodes of the decision tree. It is well suited for very large instances with many items. The reason is that SH quickly constructs a decent packing plan, whereas RLS and EA start from the empty packing plan and then cannot reach SH's performance within 10 minutes.

We show a representative excerpt of the results in Figure 3.[3] Note that we rescale the achieved objective values into the range $[0, 1]$: the objective value of packing no items at all is equivalent to 0, and the best to 1. We do this as the objective values are typically (but not always) negative. In addition, in some cases the worst value is negative and the best value positive, which makes the value ranges not suitable for a number of other visualisation techniques. The plots largely confirm our observations from the decision tree in Figure 2:

- RLS and EA perform best for a very wide range of TTP configurations.

[3] `http://cs.adelaide.edu.au/~optlog/research/ttp.php`

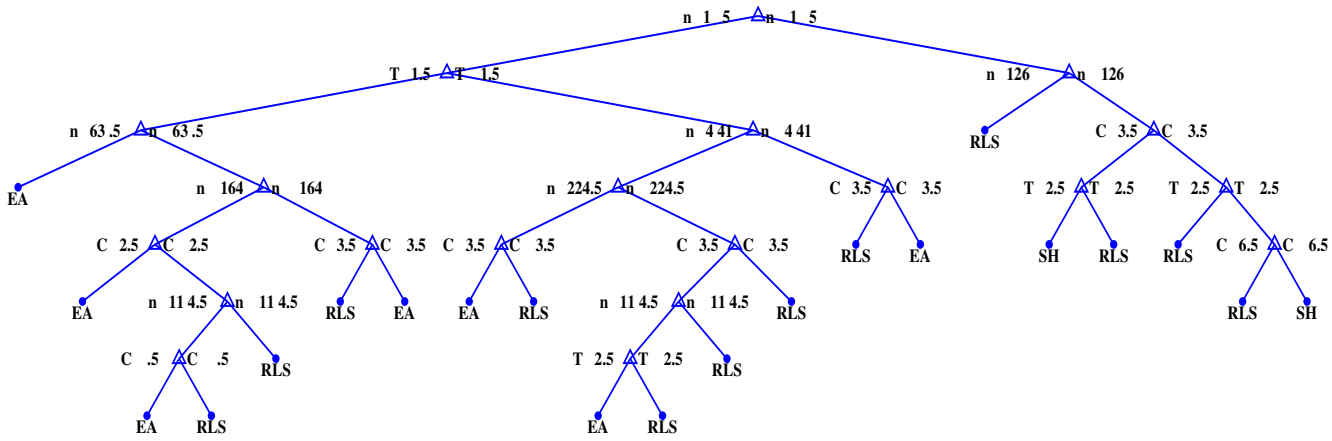

Figure 2: Decision tree based on all experimental results. n refers to the number of items, $C \in \{1, \ldots, 10\}$ to the capacity category, and T to the knapsack type. The values of $T \in \{1, 2, 3\}$ correspond to $\{bounded-strongly-corr, uncorr-similar-weights, uncorr\}$. Consequently, $T < 2.5$ is equivalent to $T \neq uncorr$.

- SH often is the best performing approach for very large TTP instances.

In addition, we notice the following interesting cases:

- For SH, the configuration of the TTP instance (what kind of items, how many, size of knapsack) seems to have a significant influence. In many cases, the achieved objective scores relative to RLS and EA are relatively stable. For example in the case of "uncorrelated item weight, $C = 10$, $F = 3$" (top right corner), all relative performances are at around 0.8. The reason for this behaviour that is relative to the starting point and relative to the performance of the iterative approaches is currently unclear.

- Compared to EA, RLS extends its good performance into larger TTP instances as it is more effective in reaching a local optimum. This can be seen in the plots as in most cases the EA marks decent before the RLS marks.

- The standard deviations for the iterative heuristics on instances with more than 5,000 cities are often quite significant. Consequently, we think that there is a lot of room left for smarter algorithms to explore the search space more efficiently.

As an example, let us have a closer look at the instances of the type "uncorrelated item weights, knapsack category 2, item factor 3" (top left plot in Figure 3). The underlying objective values are listed in Table 1 and the runtimes are shown in Figure 4. Note that we consider most of these to be "smaller" TTP instances.[4] PackNone refers to the solution where no item is packed along the tour from the Chained Lin-Kernighan algorithm; it serves as our first lower bound. SH is deterministic and we therefore do not report any standard deviations. Some observations are:

[4]Based on our observations and chosen ranges, we argue that for this F-value up to 3,000-5,000 cities result in small TTP instances.

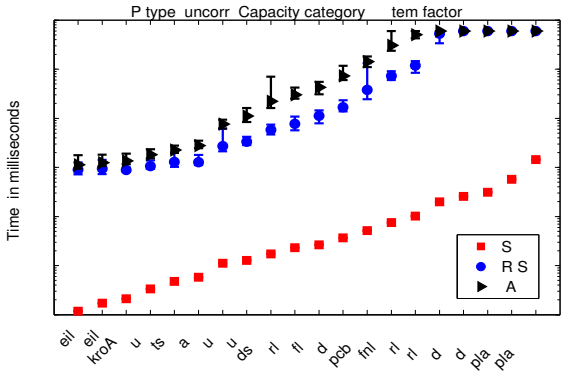

Figure 4: Subset of the results. Shown are the runtimes of the algorithms. The identifiers on the x-axis refer to the name of the underlying TSP instance.

- SH computes packing plans extremely quickly. It increases to just over 1s for the largest instance. On the other hand, the iterative RLS and EA hit the runtime limit of 10 minutes from about 6,000 cities onwards.

- It can be seen that EA's runtime is typically longer than that of RLS. The reasons are that, at times, either EA wastes time on re-evaluating solutions, or EA escapes local optima and subsequently climbs up to other optima again.

- On average, EA performs slightly better than RLS on small instances (up to 2,000-3,000 cities). RLS then clearly outperforms the other approaches on medium and larger instances. SH performs best in the case of the very largest city number.

- Iterative heuristics regularly pack items during an optimization run that are later unpacked again. The reason is that packing a heavy and invaluable item can increase the objective score temporarily during the optimization. However, as this slows down the thief, it can

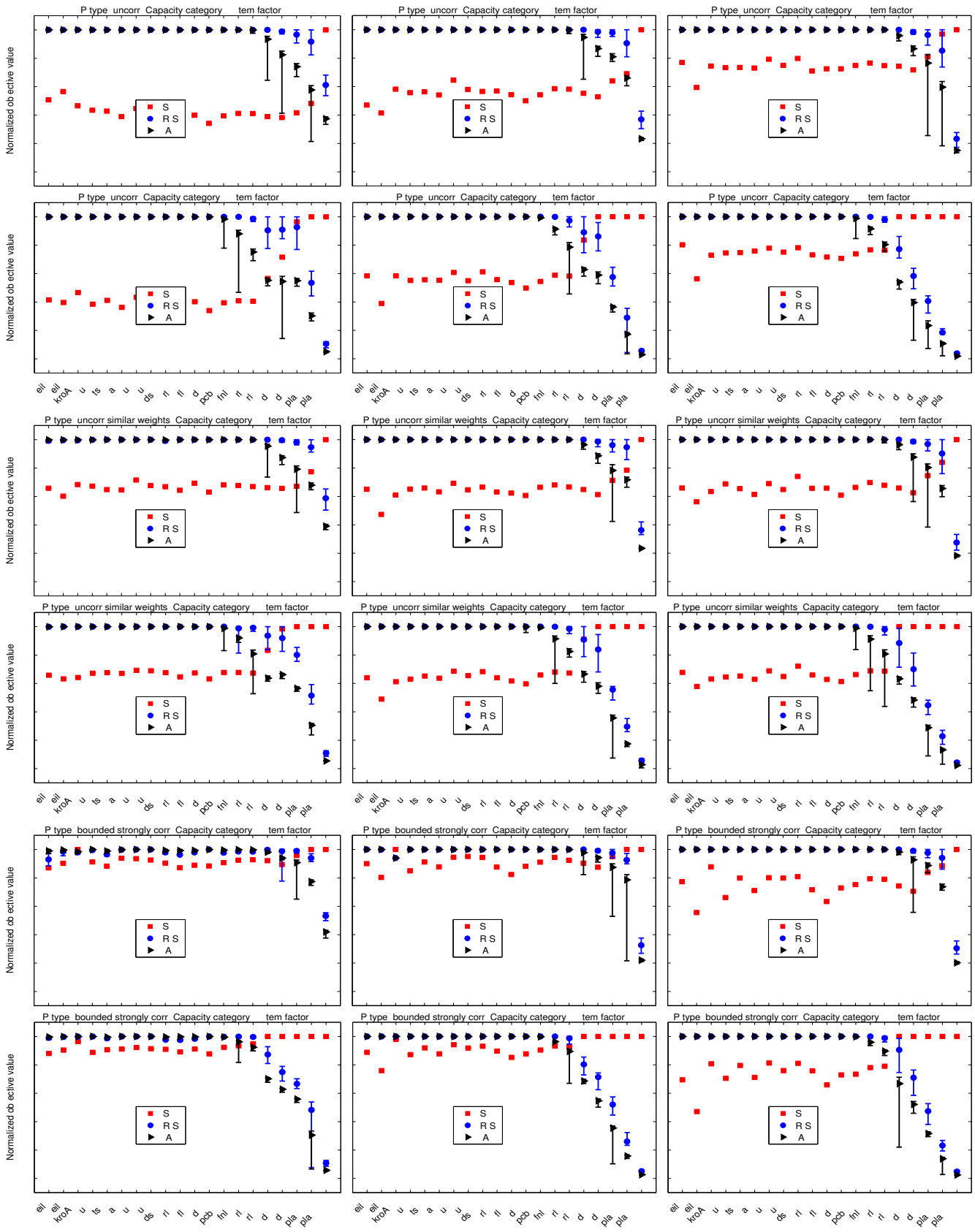

Figure 3: Subset of the results. Shown are the rescaled performances of our heuristics on a wide range of different instances. The identifiers on the x-axis refer to the name of the underlying TSP instance. Additional instance details are described in the plot titles. For example, the knapsack type is (f.t.t.b. in groups of six): uncorrelated, uncorrelated with similar weights, bounded strongly correlated.

Algorithm	eil51	eil76	kroA100	u159	ts225	a280	u574	u724	dsj1000	rl1304
RLS (mean)	8.21e3	1.16e4	1.93e4	4.03e4	5.70e4	6.32e4	1.36e5	1.67e5	1.11e5	3.12e5
RLS (std)	1.58e1	2.68e1	1.20e1	1.42e1	1.41e1	1.11e1	3.10	3.12	4.10e−1	6.23e1
EA (mean)	8.22e3	1.16e4	1.93e4	4.03e4	5.70e4	6.32e4	1.36e5	1.67e5	1.11e5	3.12e5
EA (std)	1.38e1	2.49e1	9.68	1.74e1	1.01e1	1.12e1	4.42	2.72	2.20e1	3.30e1
SH	−3.03e3	−3.65e3	−4.83e3	−5.37e3	−7.49e3	−2.05e4	−1.99e4	−5.16e4	−1.90e5	−9.69e4
PackNone	−1.46e4	−2.35e4	−2.58e4	−4.04e4	−5.57e4	−7.37e4	−1.46e5	−1.92e5	−3.75e5	−3.62e5
Algorithm	fl1577	d2103	pcb3038	fnl4461	rl5934	rl11849	d15112	d18512	pla33810	pla85900
RLS (mean)	3.57e5	4.80e5	6.50e5	9.08e5	1.44e6	2.69e6	3.44e6	3.53e6	2.02e6	−1.40e7
RLS (std)	1.66e2	1.55e2	2.21e2	1.99e2	5.11e2	1.51e3	4.96e4	1.82e5	4.61e5	6.18e5
EA (mean)	3.57e5	4.80e5	6.50e5	9.08e5	1.43e6	2.29e6	2.18e6	1.37e6	−2.16e6	−1.80e7
EA (std)	1.26e1	8.92e1	5.11e1	2.03e2	1.36e4	3.46e5	6.01e5	1.88e5	8.52e5	1.91e5
SH	−1.30e5	−2.17e5	−2.94e5	−4.23e5	−3.97e5	−1.07e6	−1.23e6	−1.74e6	−3.35e6	−7.49e6
PackNone	−4.53e5	−5.78e5	−9.09e5	−1.36e6	−1.67e6	−3.45e6	−4.17e6	−5.72e6	−9.28e6	−2.43e7

Table 1: Plot data for the instances "uncorrelated item weights, knapsack category 2, item factor 3" (see top left plot of Figure 3). Shown are the average objective values achieved and the corresponding standard deviations; the highest mean per instance is highlighted. The number of cities increases from left to right.

later become beneficial to deselect this item again due to the trade-off in knapsack renting cost vs. knapsack profit.

- The objective scores for the randomized algorithms vary hardly when the number of cities is less than 10,000. Even though different local optima are found, we conjecture that these local optima are all very close to the global optima for the chosen TSP tour.

- Interestingly, the objective values for one instance are (roughly) within one order of magnitude. This means that the packing plans have a significant influence on the TTP objective score, even though the TSP subproblem was solved almost to optimality by the used Chained Lin-Kernighan heuristic. It remains unknown by how much the achieved objective scores can be improved.

6. CONCLUSIONS

In this paper, we present a set of benchmarks for the travelling thief problem (TTP). The instances are systematically created to cover a wide range of features, and they are based on well-known instances for the travelling salesperson problem and for the knapsack problem. Our goal is to enable researches to investigate the hardness the two classical subproblems' combination. Even though we are confident that smaller instances can soon be optimally solved, we conjecture that the larger ones will remain unsolved for the years to come.

As a starting point, we provide a first set of structurally very different algorithms to effectively solve the problems. Our experiments show that a good and quick construction of a starting point is as important as an iterative search that can escape local optima.

In the future, we plan to intensively analyze the difficulty of the TTP through rigorous theoretical and experimental investigations. We plan to identify features that make instances easy or difficult to solve. The results can then be used to create problem-specific algorithms that are based on actual properties of the problem. Furthermore, we intend to discover inherent regularity in interaction between components and their mutual influence. The preliminary results have recently been obtained in [5], where the potential benefit of methods considering interdependency between components was studied.

Lastly, our instances can form the basis for many interpretations of the original TTP, such as, TTP with multiple thiefs, TTP without a restriction to visit every city, TTP with the option to sell items at specific locations, and so on.

Acknowledgements

This work was supported by ARC grants DP130104395 and DP140103400.

References

[1] Advanced generator for 0-1 Knapsack Problems. See `http://www.diku.dk/~pisinger/codes.html`.

[2] TSP Test Data. See `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html`.

[3] D. Applegate, W. J. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15:82–92, 2003.

[4] M. R. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *IEEE Congress on Evolutionary Computation*, pp. 1037–1044. IEEE, 2013.

[5] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. Socially inspired algorithms for the travelling thief problem. In *Genetic and Evolutionary Computation Conference, GECCO 2014*, 2014. to appear.

[6] I. Kara, B. Y. Kara, and M. K. Yetis. Energy minimizing vehicle routing problem. In A. Dress,

Y. Xu, and B. Zhu, editors, *Combinatorial Optimization and Applications*, Vol. 4616 of *Lecture Notes in Computer Science*, pp. 62–71. Springer Berlin Heidelberg, 2007.

[7] I. Kucukoglu, S. Ene, A. Aksoy, and N. Ozturk. Green capacitated vehicle routing problem fuel consumption optimization model. *Computational Engineering Research*, 3:16–23.

[8] C. Lin, K. Choy, G. Ho, S. Chung, and H. Lam. Survey of green vehicle routing problem: Past and future trends. *Expert Systems with Applications*, 41: 1118 – 1138, 2014.

[9] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Manage. Sci.*, 45:414–424, 1999.

[10] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3:376–384, 1991.