Exact Approaches for the Travelling Thief Problem

Junhua Wu, Markus Wagner, Sergey Polyakovskiy, and Frank Neumann

Optimisation and Logistics
School of Computer Science
The University of Adelaide, Adelaide SA, Australia

Abstract. Many evolutionary and constructive heuristic approaches have been introduced in order to solve the Travelling Thief Problem (TTP). However, the accuracy of such approaches is unknown due to their inability to find global optima. In this paper, we propose three exact algorithms and a hybrid approach to the TTP. We compare these with state-of-the-art approaches to gather a comprehensive overview on the accuracy of heuristic methods for solving small TTP instances.

1 Introduction

The travelling thief problem (TTP) [3] is a recent academic problem in which two well-known combinatorial optimisation problems interact, namely the travelling salesperson problem (TSP) and the 0-1 knapsack problem (KP). It reflects the complexity in real-world applications that contain more than one $\mathcal{NP}$-hard problem, which can be commonly observed in the areas of planning, scheduling and routing. For example, delivery problems usually consist of a routing part for the vehicle(s) and a packing part of the goods onto the vehicle(s).

Thus far, many approximate approaches have been introduced for addressing the TTP and most of them are evolutionary or heuristic [17]. Initially, Polyakovskiy, Bonyadi, Wagner, Michalewicz, and Neumann [18] proposed two iterative heuristics, namely the Random Local Search (RLS) and (1+1)-EA, based on a general approach that solves the problem in two steps, one for the TSP and one for the KP. Bonyadi et al. [4] introduced a similar two-phased algorithm named Density-based Heuristic (DH) and a method inspired by coevolution-based approaches named CoSolver. Mei et al. [11–13] also investigated the interdependency and proposed a cooperative coevolution based approach similar to CoSolver, and a memetic algorithm called MATLS that attempts to solve the problem as a whole. In 2015, Faulkner et al. [7] outperformed the existing approaches by their new operators and corresponding series of heuristics (named S1–S5 and C1–C6). Recently, Wagner [22] investigated the Max-Min Ant System (MMAS) [21] on the TTP, and El Yafrani and Ahiod [6] proposed a memetic algorithm (MA2B) and a simulated annealing algorithm (CS2SA). The results show that the new algorithms were competitive to the state-of-the-art on a different range of TTP instances. Wagner et al. [23] found in a study involving

21 approximate TTP algorithms that only a small subset of them is actually necessary to form a well-performing algorithm portfolio.

However, due to the lack of exact methods, all of the above-mentioned approximate approaches cannot be evaluated with respect to their accuracy even on small TTP instances. To address this issue, we propose three exact techniques and additional benchmark instances, which help to build a more comprehensive review of the approximate approaches.

In the remainder, we revisit the definition of the TTP in Section 2 and introduce our exact approaches in Section 3. In Section 4, we elaborate on the setup of our experiments and compare our exact and hybrid approaches with the best approximate ones. The conclusions are drawn in Section 5.

2 Problem Statement

Given is a set of cities $N = \{1, \ldots, n\}$ and a set of items $M = \{1, \ldots, m\}$. City i, $i = 2, \ldots, n$, contains a set of items $M_i = \{1, \ldots, m_i\}$, $M = \underset{i \in N}{\cup} M_i$. Item k positioned in the city i is characterised by its profit p_{ik} and weight w_{ik}. The thief must visit each of the cities exactly once starting from the first city and return back to it in the end. The distance d_{ij} between any pair of cities $i, j \in N$ is known. Any item may be selected as long as the total weight of collected items does not exceed the capacity C. A renting rate R is to be paid per each time unit taken to complete the tour. υ_{max} and υ_{min} denote the maximal and minimum speeds of the thief. Assume that there is a binary variable $y_{ik} \in \{0, 1\}$ such that $y_{ik} = 1$ iff item k is chosen in city i. The goal is to find a tour $\Pi = (x_1, \ldots, x_n)$, $x_i \in N$, along with a packing plan $P = (y_{21}, \ldots, y_{nm_n})$ such that their combination $[\Pi, P]$ maximises the reward given in the form of the following objective function.

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

where $\nu = (\upsilon_{max} - \upsilon_{min})/C$ is a constant value defined by input parameters. The minuend is the sum of all packed items' profits and the subtrahend is the amount that the thief pays for the knapsack's rent equal to the total travelling time along Π multiplied by R. In fact, the actual travel speed along the distance $d_{x_i x_{i+1}}$ depends on the accumulated weight $W_{x_i} = \sum_{j=1}^{i} \sum_{k=1}^{m_j} w_{jk} y_{jk}$ of the items collected in the preceding cities $1, \ldots, i$. This then slows down the thief and has an impact on the overall benefit Z. For a particular graph example, we refer the interested reader to [18].

3 Exact Approaches to the TTP

This section introduces three exact approaches to the TTP. Our first solution is a dynamic program (DP), which adopts the ideas applied to the simplified version of the TTP, the Packing While Travelling problem (PWT). Polyakovskiy and Neumann [17] have recently introduced the PWT, which considers a fixed tour Π and only asks for an optimal packing solution. Neumann et al. [14] show that the problem can be solved in pseudo-polynomial time via dynamic programming taking into account the fact that the weights are integer. Their dynamic programming follows the traditional scheme and sequentially examines all combinations of items and weights so that the optimal packing plan P^* can be selected among all the resulting solutions. The rest of the section first describes the DP and then continues with a branch and bound approach (BnB) and with a constraint programming (CP) technique adopted for the TTP.

3.1 Dynamic Programming

Our DP is based on the Held-Karp algorithm for the TSP [8] augmented by the dynamic programming routine [14] applied to resolve low level PWT subproblems. For a subset of cities $S \subseteq N$ and city $j \in S$, let $A(S, j, w)$ be the maximum reward of the path visiting each city in S exactly once, starting at the home city and ending at j with the total knapsack's weight of w. Obviously, if $|S| > 1$, $A(S, 1, w) = -\infty$ for any $w \in [0, C]$ since the path cannot both start and end at city 1.

Our base case consists of $A(\{1\}, 1, 0) = 0$ and of $A(\{1\}, 1, w) = -\infty$ for $0 < w \leq C$. Our general case for $A(S, j, w)$ is based on $A\left(S \setminus \{j\}, i, w - \overline{W_j}\right)$, which is the path from city 1 to city $i \in S$, plus the reward gained from visiting j right after i. In fact, i must be the best choice:

$$A(S, j, w) = \max_{i \in S : i \neq j} \left\{ A\left(S \setminus \{j\}, i, w - \overline{W_j}\left(S \setminus \{j\}, i\right)\right) + \overline{P_j}\left(S \setminus \{j\}, i\right) - \frac{d_{ij}}{\upsilon_{max} - \nu w} \right\}.$$

Here, $\overline{W}_j(S \setminus \{j\}, i)$ and $\overline{P}_j(S \setminus \{j\}, i)$ represent the total weight and the total profit of the items chosen in city j. They both result from the best solution of the PWT subproblem, where a subset of items in M_j must be optimally chosen with respect to the set of partial solutions corresponding to $A(S \setminus \{j\}, i, w)$, $w \in [0, C]$. In fact, the sub-problem considers only the items of city j and can be solved via the dynamic programming approach for the PWT [14].

We start computing solutions with all subsets of size $s = 2$ and calculate $A(S, j, w)$ sequentially for all possible knapsack's weights w and subsets $S \subseteq N$ subject to S containing city 1. We iteratively increment s and continue until $s = n$. Finally, we compute the value of an optimal solution for the complete tour as

$$\max_{i \in S : i \neq 1} \left\{ A(N, i, w) - \frac{d_{i1}}{\upsilon_{max} - \nu w} \right\}.$$

There are at most $2^n n$ subproblems, and each of them takes the time of $\mathcal{O}(n+mC)$. Therefore, the total running time is $\mathcal{O}(2^n n\,(n + mC))$. The dynamic programming is also rather expensive in terms of the memory consumption, which reaches $\mathcal{O}(2^n nC)$. In order to speed up computations, let as define an

Algorithm 1 Dynamic programming to the TTP

set $A(\{1\}, 1, 0) = 0$
for $w = 1$ to C **do**
    set $A(\{1\}, 1, w) = -\infty$
for $s = 2$ to n **do**
    for any $S \subseteq N : |S| = s,\ 1 \in S$ **do**
        for $w = 0$ to C **do**
            set $A(S, 1, w) = -\infty$
            for any $j \in S,\ j \neq 1$ **do**
                compute $A(S, j, w) =$
                $\max_{i \in S: i \neq j} \left\{ A\left(S \setminus \{j\}, i, w - \overline{W_j}\left(S \setminus \{j\}, i\right)\right) + \overline{P_j}\left(S \setminus \{j\}, i\right) - \frac{d_{ij}}{\upsilon_{max} - \nu w} \right\}$
return $\max_{i \in S: i \neq 1} \left\{ A(N, i, w) - \frac{d_{i1}}{\upsilon_{max} - \nu w} \right\}$

upper bound on the value of a feasible solution that can be derived from the partial solutions corresponding to $A(S \setminus \{j\}, i, w)$, for any $w \in [0, C]$, as follows:

$$E_U(A(S, j, \cdot)) = \max_{0 \leq w \leq W} A(S, j, w) + \sum_{k \in N \setminus S} \sum_{l=1}^{m_k} p_{kl} - R \frac{d_{j1}}{\upsilon_{max}}$$

It estimates the maximal profit that the thief may obtain by passing the remaining part of the tour with the maximal speed; that is, generating the minimal possible cost of traveling. Obviously, an optimal solution must not exceed this bound. Therefore, if an incumbent solution $Z([\Pi', P'])$ exists, those partial solutions whose $E_U(A(S, j, w)) < Z([\Pi', P'])$ can be ignored. In practice, we can obtain an incumbent solution in two stages. First, a feasible solution Π' for the TSP part of the problem can be computed by a TSP solver such as Concorde [1] or by the Lin-Kernighan algorithm [10]. Second, the dynamic programming for PWT can be applied to determine the best packing plan P' for Π'.

3.2 Branch and Bound Search

Now, we introduce a branch and bound search for the TTP employing the upper bound E_U defined in Section 3.1. Algorithm 2 depicts the pseudocode, where $\Pi_i, i \in \{1, ..., n\}$ denotes a sub-permutation of Π with the cities 1 to i visited, and f_i is the mapping $f : w \mapsto P$ calculated for Π by the dynamic programming for the PWT.

A way to tighten the upper bound E_U is by providing a better estimation of the remaining distance from the current city k to the last city of the tour. Currently, the shortest distance from k to 1, i.e. d_{k1}, is used. The following two ways can improve the estimation: (i) the use of distance d_{f1} from city f to city 1, where f is the farthest unvisited city from 1; (ii) the use of distance $d^* - d_t$, where d^* is the shortest path that can be pre-calculated and d_t is the distance passed so far to achieve city k in the tour Π. These two ideas can be joined together by using the $\max\{d_f, (d^* - d_t)\}$ to enhance the result.

Algorithm 2 Branch and Bound Search for the TTP

```
procedure BnB Search
    Create an initial solution to gain the benefit best and an tour permutation Π
    Create an empty mapping M
    Set l = 0
    Search(Π, l, M, best)
function Search(Π, l, M, best)
    if l == n then
        calculate Z([Π, f_n(·)]) from Z([Π_{n−1}, f_{n−1}(·)]) in M
        return max{max Z([Π, f_n(·)]), best}
    else
        for i = l + 1 to n do
            Swap cities l + 1 and i in Π
            Set M' = Calculate Z([Π_{l+1}, f_{l+1}(·)]) from Z([Π_l, f_l(·)]) in M
            if max E_U([Π_{l+1}, f_{l+1}(·)]) > best then
                best = max{best, Search(Π, l + 1, M′, best) }
            Swap cities l + 1 and i in Π
        return best
```

3.3 Constraint Programming

Now, we present our third exact approach adopting the existing state-of-the-art constraint programming (CP) paradigm [9]. Our model employs a simple permutation based representation of the tour which allows the use of the `AllDifferent` filtering algorithm [2]. Similarly to the Section 2, a vector $W = (W_1, \dots, W_n)$ is used to refer to the total weights accumulated in the cities of tour Π. Specifically, W_i is the weight of the knapsack when the thief departs from city i. The model bases the search on two types of decision variables:

- $\boldsymbol{x}$ denotes the particular positions of the cities in tour Π. Variable x_i takes the value of $j \in N$ to indicate that j is the ith city to be visited. The initial variable domain of x_1 is $D(x_1) = \{1\}$ and it is $D(x_i) = N \setminus \{i\}$ for any subsequently visited city $i = 2, \dots, n$.
- $\boldsymbol{y}$ signals on the selection of an item in the packing plan P. Variable y_{ik}, $i \in N$, $k \in M_i$, is binary, therefore $D(y_{ik}) = \{0, 1\}$.

Furthermore, an integer-valued vector d is used to express the distance matrix so that its element $n(x_i - 1) + x_{i+1}$ equals the distance $d_{x_i x_{i+1}}$ between two consecutive cities x_i and x_{i+1} in Π.

The model relies on the `AllDifferent`$[x_1, \dots, x_n]$ constraint, which ensures that the values of $x_1, \dots, x_n$ are distinct. It also involves the `Element`(g, h) expression, which returns the hth variable in the list of variables g. In total, the model (CPTTP) consists of the objective function and constraints as depicted in the Figure (1). Expression (4) calculates the objective value according to function (1). Constraint (5) verifies that all the cities are assigned to different positions, and thus are visited exactly once. This is a sub-tour elimination con-

$$max \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{ij} y_{ij}$$

$$- R\left(\sum_{i=1}^{n-1} \frac{\texttt{Element}(d, n\,(x_i - 1) + x_{i+1})}{v_{max} - \nu\texttt{Element}(W, x_i)} + \frac{\texttt{Element}(d, n\,(x_n - 1) + 1)}{v_{max} - \nu\texttt{Element}(W, x_n)}\right) \quad (2)$$

$$\texttt{AllDifferent}[x_1, \ldots, x_n] \quad (3)$$

$$W_i = W_{i-1} + \sum_{j \in M_i} w_{ij} y_{ij},\ i \in \{2, \ldots, n\} \quad (4)$$

$$W_n \leq C \quad (5)$$

Fig. 1. Constraint programming model to the TTP

straint. Equation (6) calculates the weight W_i of all the items collected in the cities $1, \ldots, i$. Equation (7) is a capacity constraint.

The performance of a CP model depends on its solver; specifically, on the filtering algorithms and on the search strategies it applies. Here, we use IBM ILOG CP Optimiser 12.6.2 with its searching algorithm set to the *restart mode*. This mode adopts a general purpose search strategy [19] inspired from integer programming techniques and is based on the concept of the impact of a variable. The impact measures the importance of a variable in reducing the search space. The impacts, which are learned from the observation of the domains' reduction during the search, help the restart mode dramatically improve the performance of the search. Within the search, the cities are assigned to the positions first and then the items are decided on. Therefore, the solver instantiates $x_1, \ldots, x_n$ prior to $y_{21}, \ldots, y_{nm_n}$ variables applying its default selection strategy. Our extensive study shows that such an order gives the best results fast.

4 Computational Experiments

In this section, we first compare the performance of the exact approaches to TTP in order to find the best one for setting the baseline for the subsequent comparison of the approximate approaches. Our experiments run on the CPU cluster of the Phoenix HPC at the University of Adelaide, which contains 3072 Intel(R) Xeon(R) 2.30GHz CPU cores and 12TB of memory. We allocate one CPU core and 32GB of memory to each individual experiment.

4.1 Computational Set Up

To run our experiments, we generate an additional set of small-sized instances following the way proposed in [18][1]. We use only a single instance of the original

[1] All instances are available online:
http://cs.adelaide.edu.au/ optlog/research/ttp.php

Instance	n	m	Running time (in seconds)		
			DP	BnB	CP
eil51_n05_m4_uncorr_01	5	4	0.018	0.023	0.222
eil51_n06_m5_uncorr_01	6	5	0.07	0.079	0.24
eil51_n07_m6_uncorr_01	7	6	0.143	0.195	0.497
eil51_n08_m7_uncorr_01	8	7	0.343	0.505	4.594
eil51_n09_m8_uncorr_01	9	8	0.633	1.492	63.838
eil51_n10_m9_uncorr_01	10	9	0.933	5.188	776.55
eil51_n11_m10_uncorr_01	11	10	2.414	23.106	12861.181
eil51_n12_m11_uncorr_01	12	11	3.938	204.786	-
eil51_n13_m12_uncorr_01	13	12	14.217	2007.074	-
eil51_n14_m13_uncorr_01	14	13	13.408	36944.146	-
eil51_n15_m14_uncorr_01	15	14	89.461	-	-
eil51_n16_m15_uncorr_01	16	15	59.526	-	-
eil51_n17_m16_uncorr_01	17	16	134.905	-	-
eil51_n18_m17_uncorr_01	18	17	366.082	-	-
eil51_n19_m18_uncorr_01	19	18	830.18	-	-
eil51_n20_m19_uncorr_01	20	19	2456.873	-	-

Table 1. Columns 'n' and 'm' denote the number of cities and the number of items, respectively. Running times are given in seconds for DP, BnB and CP for different numbers of cities and items. '-' denotes the case when an approach failed to achieve an optimal solution in the given time limit.

TSP library [20] as the starting point for our new subset. It is entitled as `eil51` and contains 51 cities. Out of these cities, we select uniformly at random cities that we removed in order to obtain smaller test problems with $n = 5, \ldots, 20$ cities. To set up the knapsack component of the problem, we adopt the approach given in [16] and use the corresponding problem generator available in [15]. As one of the input parameters, the generator asks for the range of coefficients, which we set to 1000. In total, we create knapsack test problems containing $k(n-1)$, $k \in \{1, 5, 10\}$ items and which are characterised by a knapsack capacity category $Q \in \{1, 6, 10\}$. Our experiments focus on *uncorrelated* (`uncorr`), *uncorrelated with similar weights* (`uncorr-s-w`), and *multiple strongly correlated* (`m-s-corr`) types of instances. At the stage of assigning the items of a knapsack instance to the particular cities of a given TSP tour, we sort the items in descending order of their profits and the second city obtains k, $k \in \{1, 5, 10\}$, items of the largest profits, the third city then has the next k items, and so on. All the instances use the "CEIL_2D" for intra-city distances, which means that the Euclidean distances are rounded up to the nearest integer. We set υ_{min} and υ_{max} to 0.1 and 1.

Tables 1 and 3 illustrate the results of the experiments. The test instances' names should be read as follows. First, `eil51` stays for the name of the original TSP problem. The values succeeding n and m denote the actual number of cities and the total number of items, respectively, which are further followed by the generation type of a knapsack problem. Finally, the postfixes 1, 6 and, 10 in the instances' names describe the knapsack's capacity C.

4.2 Comparison of the exact approaches

We compare the three exact algorithms by allocating each instance a generous 24-hour time limit. Our aim is to analyse the running time of the approaches influenced by the increasing number of cities. Table 1 shows the running time of the approaches.

4.3 Comparison between DP and Approximate Approaches

With the exact approaches being introduced, approximate approaches can be evaluated with respect to their accuracy to the optima. In the case of the TTP, most state-of-the-art approximate approaches are evolutionary algorithms and local searches, such as Memetic Algorithm with 2-OPT and Bit-flip (MA2B), CoSolver-based with 2-OPT, and Simulated Annealing (CS2SA) in [6], CoSolver-based with 2-OPT and Bit-flip (CS2B) in [5], and S1, S5, and C5 in [7].

Hybrid Approaches. In addition to existing heuristics, we introduce enhanced approaches of S1 and S5, which are hybrids of the two and that one of dynamic programming for the PWT [14]. The original S1 and S5 work as follows. First, a single TSP tour is computed using the Chained Lin-Kernighan-Heuristic [10], then a fast packing heuristic is applied. S1 performs these two steps only once and only in this order, while S5 repeats S1 until the time budget is exhausted. Our hybrids DP-S1 and DP-S5 are equivalent to these two algorithms, however, they use the exact dynamic programming to the PWT as a packing solver. This provides better results as we can now compute the optimal packing for the sampled TSP tours.

Results. We start by showing a performance summary of 10 algorithms on 432 instances in Table 2. In addition, Table 3 shows detailed results for a subset of the best approaches on a subset of instances. Figure 2 shows the results of the entire comparison. We include trend lines[2] for two selected approaches, which we will explain in the following.

We would like to highlight the following observations:

1. S1 performs badly across a wide range of instances. Its restart variant S5 performs better, however, its lack of a local search becomes apart in its relatively bad performance (compared to other approaches) on small instances.
2. C5 performs better than both S1 and S5, which is most likely due to its local searches that differentiate it from S1 and S5. Still, we can see a "hump" in its trend line for smaller instances, which flattens out quickly for larger instances.
3. The dynamic programming variants DP-S1 and DP-S5 perform slightly better than S1 and S5, which shows the difference in quality of the packing strategy; however, this is at times balanced out by the faster packing which

[2] They are fitted polynomials of degree six used only for visualisation purposes.

gap	MA2B	CS2B	CS2SA	S1	S5	C5	DP-S1	DP-S5
avg	0.3%	15.3%	11.5%	38.9%	15.7%	09.9%	30.1%	3.3%
stdev	2.2%	17.8%	16.7%	29.4%	24.6%	18.8%	20.1%	8.5%
$\#_{\mathrm{opt}}$	312	70	117	3	42	193	5	85
$\#_{1\%}$	265	100	132	10	160	193	9	245
$\#_{10\%}$	324	161	206	27	203	240	33	288

Table 2. Performance summary of heuristic TTP solvers across all instances for which the optimal result has been obtained. $\#_{\mathrm{opt}}$ is the number of times when the average of 10 independent repetitions is equal to the optimum. $\#_{1\%}$ and $\#_{10\%}$ show the number of times the averages are within 1% and 10%.

allows more TSP tours to be sampled. For small instances, DP-S5 lacks a local search on the tours, which is why its gap to the optimum is relatively large, as shown by the respective trend lines.

4. MA2B dominates the field with outstanding performance across all instances, independent of number of cities and number of items. Remarkable is the high reliability with which it reaches a global optimum.

Interestingly, all approaches seem to have difficulties solving instances with the knapsack configuration multiple-strongly-corr_01 (see Table 3). Compared to the other two knapsack types, TTP-DP takes the longest to solve the strongly correlated ones. Also, these tend to be the only instances for which the heuristics rarely find optimal solutions, if at all.

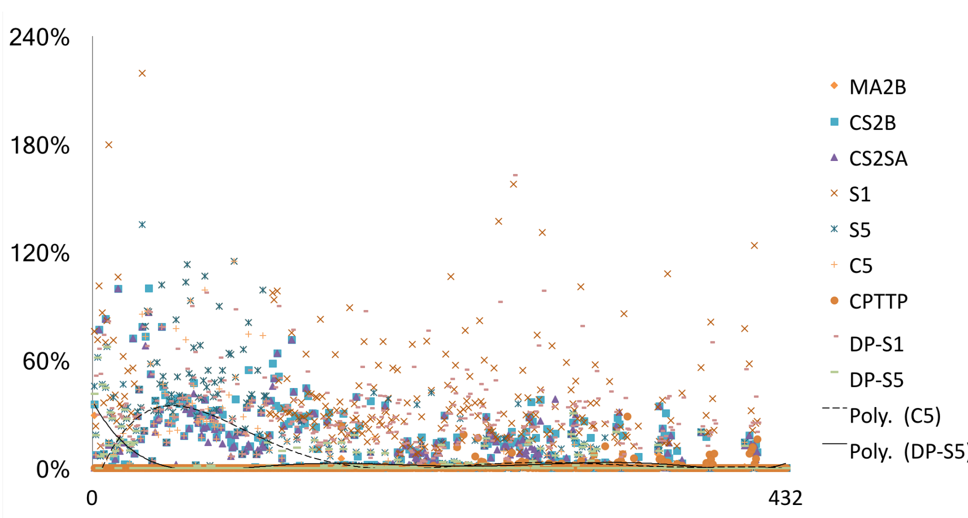

Fig. 2. Showing a gap to an optimal solution when one has been obtained by an exact approach. From left to right: the 432 instances first sorted by the number of cities, then by the total number of items.

	TTP-DP		MA2B			C5		DP-S5	
Instance	OPT	RT	Gap	Std	RT	Gap	Std	Gap	Std
eil51_n05_m4_multiple-strongly-corr_01	619.227	0.02	29.1	12.1	2.71	35.5	1.20e-6	41.3	0.0
eil51_n05_m4_uncorr_01	466.929	0.02	0.0	0.0	3.22	0.0	2.20e-6	0.0	2.20e-6
eil51_n05_m4_uncorr-similar-weights_01	299.281	0.02	0.0	0.0	3.21	7.8	2.40e-6	7.8	1.20e-6
eil51_n05_m20_multiple-strongly-corr_01	773.573	0.08	13.4	0.0	1.44	14.3	0.0	12.8	0.0
eil51_n05_m20_uncorr_01	2144.796	0.07	0.0	0.0	3.35	7.4	0.0	6.6	2.30e-6
eil51_n05_m20_uncorr-similar-weights_01	269.015	0.04	0.0	0.0	3.51	0.0	2.30e-6	0.0	0.0
eil51_n10_m9_multiple-strongly-corr_01	573.897	1.21	0.0	0.0	6.07	0.0	0.0	0.0	0.0
eil51_n10_m9_uncorr_01	1125.715	0.93	0.0	0.0	6.06	0.0	1.30e-6	0.0	1.30e-6
eil51_n10_m9_uncorr-similar-weights_01	753.230	0.86	0.0	0.0	5.87	0.0	0.0	0.0	0.0
eil51_n10_m45_multiple-strongly-corr_01	1091.127	14.89	0.0	0.0	7.99	0.0	0.0	0.0	0.0
eil51_n10_m45_uncorr_01	6009.431	6.39	0.0	0.0	8.6	6.6	2.30e-6	0.0	0.0
eil51_n10_m45_uncorr-similar-weights_01	3009.553	8.87	0.0	0.0	6.78	0.0	2.30e-6	0.0	2.30e-6
eil51_n12_m11_multiple-strongly-corr_01	648.546	4.58	0.0	0.0	6.08	4.6	2.20e-6	4.6	2.20e-6
eil51_n12_m11_uncorr_01	1717.699	3.94	0.0	0.0	7.21	0.0	1.20e-6	0.0	1.20e-6
eil51_n12_m11_uncorr-similar-weights_01	774.107	3.36	0.0	0.0	7.03	0.0	2.30e-6	0.0	2.30e-6
eil51_n12_m55_multiple-strongly-corr_01	1251.780	117.99	0.0	0.0	9.19	0.0	0.0	0.0	0.0
eil51_n12_m55_uncorr_01	8838.012	35.79	0.0	0.0	9.76	0.0	0.0	0.0	0.0
eil51_n12_m55_uncorr-similar-weights_01	3734.895	38.36	12.3	0.0	8.34	12.3	0.0	0.2	0.0
eil51_n15_m14_multiple-strongly-corr_01	547.419	39.82	0.0	0.0	7.87	14.1	1.30e-6	13.3	1.30e-6
eil51_n15_m14_uncorr_01	2392.996	89.46	0.0	0.0	7.28	3.8	0.0	3.8	0.0
eil51_n15_m14_uncorr-similar-weights_01	637.419	16.35	0.0	0.0	6.86	0.0	1.60e-6	0.0	1.60e-6
eil51_n15_m70_multiple-strongly-corr_01	920.372	3984.29	2.1	1.1	12.11	0.0	2.70e-6	0.0	2.70e-6
eil51_n15_m70_uncorr_01	9922.137	740.22	0.0	0.0	9.67	7	1.20e-6	1.9	0.0
eil51_n15_m70_uncorr-similar-weights_01	4659.623	867.78	0.0	0.0	7.98	0.0	0.0	0.0	0.0
eil51_n16_m15_multiple-strongly-corr_01	794.745	105.5	0.0	0.0	7.7	18.9	1.6e-6	18.9	1.6e-6
eil51_n16_m15_multiple-strongly-corr_10	4498.848	623.4	0.0	0.0	9.1	12.9	0.0	16.6	1.3e-6
eil51_n16_m15_uncorr_01	2490.889	59.5	1.0	0.7	8.4	1.6	2.3e-6	1.6	2.3e-6
eil51_n16_m15_uncorr_10	3601.077	211.5	0.0	0.0	9.0	7.1	1.6e-6	7.1	1.6e-6
eil51_n16_m15_uncorr-similar-weights_01	540.897	36.4	0.0	0.0	8.5	0.0	3.0e-6	0.0	3.0e-6
eil51_n16_m15_uncorr-similar-weights_10	3948.211	245.4	0.0	0.0	8.7	5.8	1.5e-6	13.6	0.0
eil51_n17_m16_multiple-strongly-corr_01	685.565	248.6	0.0	0.0	8.4	0.2	1.5e-6	0.0	1.5e-6
eil51_n17_m16_multiple-strongly-corr_10	3826.098	2190.4	0.0	0.0	9.8	0.0	1.5e-6	0.0	1.5e-6
eil51_n17_m16_uncorr_01	2342.664	134.9	0.0	0.0	8.3	0.0	0.0	0.0	0.0
eil51_n17_m16_uncorr_10	2275.279	554.5	0.0	0.0	9.6	0.0	0.0	0.0	0.0
eil51_n17_m16_uncorr-similar-weights_01	556.851	70.8	0.0	0.0	8.1	0.0	0.0	0.0	0.0
eil51_n17_m16_uncorr-similar-weights_10	2935.961	787.7	0.0	0.0	9.7	0.0	0.0	0.0	0.0
eil51_n18_m17_multiple-strongly-corr_01	834.031	715.7	7.9	0.8	10.2	9.2	0.0	12.9	1.7e-6
eil51_n18_m17_multiple-strongly-corr_10	5531.373	6252.4	0.0	0.0	10.5	0.4	1.5e-6	0.4	1.5e-6
eil51_n18_m17_uncorr_01	2644.491	366.1	0.0	0.0	9.7	0.2	0.0	1.8	0.0
eil51_n18_m17_uncorr_10	3222.603	1462.7	0.0	0.0	10.3	0.0	1.3e-6	0.2	0.0
eil51_n18_m17_uncorr-similar-weights_01	532.906	148.3	0.0	0.0	8.5	0.0	1.3e-6	0.0	1.3e-6
eil51_n18_m17_uncorr-similar-weights_10	4420.438	1929.3	0.0	0.0	9.9	0.0	2.9e-6	0.3	1.8e-6
eil51_n19_m18_multiple-strongly-corr_01	910.229	1771.6	0.0	0.0	9.3	20.1	1.6e-6	20.1	1.6e-6
eil51_n19_m18_multiple-strongly-corr_10	-	-	-	-	10.4	-	-	-	-
eil51_n19_m18_uncorr_01	2604.844	830.2	0.0	0.0	9.7	0.0	0.0	0.0	0.0
eil51_n19_m18_uncorr_10	4048.408	3884.3	0.0	0.0	10.9	0.0	1.4e-6	0.0	1.4e-6
eil51_n19_m18_uncorr-similar-weights_01	472.186	412.3	0.0	0.0	9.2	0.0	1.5e-6	0.0	1.5e-6
eil51_n19_m18_uncorr-similar-weights_10	5573.695	5878.8	0.0	0.0	10.5	0.0	0.0	0.0	0.0
eil51_n20_m19_multiple-strongly-corr_01	518.189	4533.7	0.6	0.6	11.1	14.1	1.4e-6	12.3	0.0
eil51_n20_m19_multiple-strongly-corr_10	-	-	-	-	12.1	-	-	-	-
eil51_n20_m19_uncorr_01	2092.673	2456.9	0.0	0.0	8.7	0.0	0.0	0.0	0.0
eil51_n20_m19_uncorr_10	3044.391	12776.0	0.0	0.0	9.8	0.0	0.0	0.0	0.0
eil51_n20_m19_uncorr-similar-weights_01	451.052	1007.7	0.0	0.0	7.9	0.0	0.0	0.0	0.0
eil51_n20_m19_uncorr-similar-weights_10	4169.799	15075.7	0.0	0.0	9.4	0.0	0.0	0.0	0.0

Table 3. Comparison between DP and the approximate approaches running in 10 minutes limits. Each approximate algorithm runs 10 times for each instance and use the average as the objective Obj. Gap is measured by $\frac{OPT-Obj}{OPT}\%$ and runtime (RT) is in second. The results of C5 and DP-S5 are obtained when they reach the time limit of 10 minutes per instance. Highlighted in blue are the best approximate results. DP runs out of memory for the instances without results.

5 Conclusion

The travelling thief problem (TTP) has attracted significant attention in recent years within the evolutionary computation community. In this paper, we have presented and evaluated exact approaches for the TTP based on dynamic programming, branch and bound, and constraint programming. We have used the exact solutions provided by our DP approach to evaluate the performance of current state-of-the-art TTP solvers. Our investigations show that they are obtaining in most cases (close to) optimal solutions. However, for a small fraction of tested instances we obverse a gap to the optimal solution of more than 10%.

Acknowledgements

This work was supported by the Australian Research councils through grants DP130104395 and DE160100850, and by the supercomputing resources provided by the Phoenix HPC service at the University of Adelaide.

Bibliography

[1] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde tsp solver. *http://www.math.uwaterloo.ca/tsp/concorde.html*, 2006.

[2] P. Benchimol, W.-J. v. Hoeve, J.-C. Régin, L.-M. Rousseau, and M. Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3):205–233, Jul 2012. ISSN 1572-9354. doi: 10.1007/s10601-012-9119-x.

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

[4] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 421–428. ACM, 2014.

[5] M. El Yafrani and B. Ahiod. Cosolver2b: An efficient local search heuristic for the travelling thief problem. In *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*, pages 1–5. IEEE, 2015.

[6] M. El Yafrani and B. Ahiod. Population-based vs. single-solution heuristics for the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 317–324. ACM, 2016.

[7] H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner. Approximate approaches to the traveling thief problem. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 385–392. ACM, 2015.

[8] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, pages 71.201–71.204. ACM, 1961.
[9] J. N. Hooker. Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14(4):295 – 321, 2002.
[10] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.
[11] Y. Mei, X. Li, and X. Yao. *Improving Efficiency of Heuristics for the Large Scale Traveling Thief Problem*, pages 631–643. Springer International Publishing, Cham, 2014. ISBN 978-3-319-13563-2. doi: 10.1007/978-3-319-13563-2˙53.
[12] Y. Mei, X. Li, F. Salim, and X. Yao. Heuristic evolution with genetic programming for traveling thief problem. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2753–2760, May 2015. doi: 10.1109/CEC.2015.7257230.
[13] Y. Mei, X. Li, and X. Yao. On investigation of interdependence between sub-problems of the travelling thief problem. *Soft Computing*, 20(1):157–172, 2016.
[14] F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu. A Fully Polynomial Time Approximation Scheme for Packing While Traveling. *ArXiv e-prints*, 2017.
[15] D. Pisinger. Advanced Generator for 0-1 Knapsack Problem. See http://www.diku.dk/∼pisinger/codes.html.
[16] D. Pisinger. Where are the hard knapsack problems? *Comput. Oper. Res.*, 32(9):2271–2284, Sept. 2005. ISSN 0305-0548. doi: 10.1016/j.cor.2004.03.002.
[17] S. Polyakovskiy and F. Neumann. The packing while traveling problem. *European Journal of Operational Research*, 258(2):424 – 439, 2017.
[18] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 477–484. ACM, 2014.
[19] P. Refalo. *Principles and Practice of Constraint Programming – CP 2004*, chapter Impact-Based Search Strategies for Constraint Programming, pages 557–571. Springer, 2004.
[20] G. Reinelt. TSPLIB- a traveling salesman problem library. *ORSA Journal of Computing*, 3(4):376–384, 1991.
[21] T. Stützle and H. H. Hoos. MAX MIN Ant System. *Future Generation Computer Systems*, 16(8):889 – 914, 2000.
[22] M. Wagner. Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle, editors, *Swarm Intelligence: 10th International Conference, ANTS 2016, Brussels, Belgium, September 7-9, 2016, Proceedings*, pages 273–281. Springer, 2016.
[23] M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, pages 1–26, 2017.