A Cooperative Coordination Solver for Travelling Thief Problems

Majid Namazi 1,2, Conrad Sanderson 1,2, M.A. Hakim Newton 2, Abdul Sattar 2

1 Data61 / CSIRO, Australia
2 Griffith University, Australia

Abstract

In the travelling thief problem (TTP), a thief undertakes a cyclic tour through a set of cities, and according to a picking plan, picks a subset of available items into a rented knapsack with limited capacity. The overall aim is to maximise profit while minimising renting cost. TTP thus combines two interdependent NP-hard components: the travelling salesman problem (TSP) and the knapsack problem (KP). Existing approaches for TTP typically solve the TSP and KP components in an interleaved fashion, where the solution of one component is held fixed while the solution of the other component is changed. This indicates poor coordination between solving the two components, which may lead to poor quality TTP solutions. The 2-OPT heuristic is often used for solving the TSP component, which reverses a segment in the cyclic tour. Within the TTP context, the 2-OPT heuristic does not take into account the picking plan, which can result in a lower objective value. This in turn can result in the tour modification to be rejected by a solver. To address this issue, we propose an extended form of the 2-OPT heuristic in order to change the picking plan in coordination with modifying the tour. Items deemed as less profitable and picked in cities earlier in the reversed segment are replaced by items that tend to be equally or more profitable and not picked in cities later in the reversed segment. The picking plan is further changed through a modified form of the hill-climbing bit-flip search, where changes in the picking state are only permitted for boundary items, which are defined as lowest profitable picked items or highest profitable unpicked items. This restriction reduces the amount of time spent on the KP component, thereby allowing more tours to be evaluated by the TSP component within a given time budget. The two modified heuristics form the basis of a new cooperative coordination solver, which is shown to outperform several state-of-the-art TTP solvers on a broad range of benchmark TTP instances.1

Keywords: multi-component optimisation; interdependent components; travelling thief problem; travelling salesman problem; knapsack problem.

1. Introduction

Real-world constraint optimisation problems [2], such as supply chain management, often consist of multiple interdependent components [3]. This interdependency makes solving these problems very challenging: finding an optimal solution to each component separately does not guarantee finding an optimal solution to the whole problem [4, 5]. The travelling thief problem (TTP) [3, 6] combines two interdependent NP-hard components: the travelling salesman problem (TSP) [7] and the knapsack problem (KP) [8]. In TTP, a thief makes a cyclic tour through a set of given cities and using a picking plan, picks a subset of available items into a rented knapsack with limited capacity. As items are picked up at each subsequent city, the total profit and weight of the items in the knapsack increases, while the speed of the thief decreases, thereby increasing the total travelling time and hence the cost of renting the knapsack. The overall goal in TTP is to simultaneously maximise the total profit of the picked items and minimise the renting cost. TTP can be thought of as a proxy for many real-world logistics problems [9].

Many TTP solvers typically solve the TSP and KP components in an interleaved fashion, where the two components are solved using two separate modules [6]. In this context, the aim of solving the TSP component is to minimise the total travelling time given a fixed picking plan, while the aim of solving the KP component is to maximise the total profit given a fixed cyclic tour. However, keeping the solution of one component fixed while the solution of the other component is changed indicates poor coordination between solving the two components, which may lead to poor quality TTP solutions.

The 2-OPT segment reversing heuristic [10] is often used for solving the TSP component. Within the TTP context, the 2-OPT heuristic does not take into account the picking plan, which can result in a decrease of the TTP objective value. This in turn can result in the tour modification to be rejected, which suggests that many possible segment reversals can be rejected without considering potential changes to the picking plan. This is an inefficient search of the TTP solution space.

1This article is a revised and extended version of our earlier work [1].
For solving the KP component, a popular approach is the bit-flip search [6, 11], which is a hill-climber that searches via flipping the picking status (from picked to unpicked and vice-versa) of one item at a time. The downside of this approach is that the small and untargeted change in the picking plan results in a slow and meandering exploration of the solution space. When a time limit is placed for finding a solution, the solution space may not be explored adequately, which can contribute to poor quality TTP solutions.

To address the lack of coordination between solving the TSP and KP components, we propose an extended and modified form of the 2-OPT heuristic, termed Profit Guided Coordination Heuristic (PGCH), which explicitly adjusts the picking plan in coordination with changes made to the cyclic tour. After reversing the segment, items deemed as less profitable and picked in cities earlier in the reversed segment are replaced by items that tend to be equally or more profitable and not picked in cities later in the reversed segment. We also propose a more targeted form of the bit-flip search, termed Boundary Bit-Flip search, where a restriction is placed to only consider changes to the picking state of boundary items. We define two types of boundary items: (i) lowest profitable picked items among all items picked earlier, and (ii) highest profitable unpicked items among all items not picked later. This reduces the amount of time spent on the KP component, thereby allowing more tours to be evaluated by the TSP component within a time limit. We combine the proposed PGCH and Boundary Bit-Flip approaches into a new TTP solver, termed as cooperative coordination (CoCo) solver. Comparative evaluations on a broad range of benchmark TTP instances indicate that the proposed solver outperforms several state-of-the-art TTP solvers: MATLS [9], S5 [11] and CS2SA* [12].

We continue the paper as follows. Section 2 provides an overview of related work. Section 3 formally defines TTP, the 2-OPT heuristic, and the bit-flip operator. Section 4 describes the proposed coordination heuristic. Section 5 describes the proposed targeted form of bit-flip search. Section 6 combines the two proposed heuristics into the proposed CoCo solver. Section 7 provides the comparative evaluation. Section 8 summarises the main findings.

2. Related Work

TTP was introduced in [3] with many benchmark instances given in [6]. Existing TTP solvers can be grouped into 5 main categories: (i) constructive methods, (ii) fixed-tour methods, (iii) cooperative methods, (iv) full encoding methods, and (v) hyper-heuristic methods. Each of the categories is briefly overviewed below. For a more thorough treatment, the reader is directed to the recent review of TTP solvers in [13].

In constructive methods, an initial cyclic tour is generated for the TSP component using the classic Chained LinKernighan heuristic [14]. The tour is then kept fixed while the picking plan for the KP component is computed by using scores assigned to the items based on their profit, weight and position in the tour. This category includes methods such as Simple Heuristic [6], Density-based Heuristic [15], Insertion [9] and Packiterative [11]. These methods are used in restart-based algorithms such as S5 [11] and in the initialisation phase of more complex methods.

In fixed-tour methods, after generating an initial cyclic tour as per constructive methods, an iterative improvement heuristic is used to solve the KP component. Two iterative methods for solving the KP component are proposed in [6]: (i) Random Local Search, which is a hill-climbing bit-flip search where the picking status of a randomly selected item is flipped in each iteration, and (ii) (1+1)-EA, a simple evolutionary algorithm where the picking status of a set of randomly selected items is flipped in each iteration.

Cooperative methods are iterative approaches based on co-operational co-evolution [16]. After generating an initial TTP solution using a constructive or fixed-tour method, the TSP and KP components are solved by two separate modules. These two modules are executed by a coordinating agent (or meta-optimiser) in an interleaved form. The coordinating agent combines the two solutions to produce an overall solution, thereby considering the interdependency between the TSP and KP components [13]. Example methods include CoSolver [15], CoSolver with 2-OPT and Simulated Annealing (CS2SA) [17], and CS2SA with offline instance-based parameter tuning (CS2SA*) [12].

In full-encoding methods, the problem is considered as a whole. Example methods include Memetic Algorithm with Two-stage Local Search (MATLS) [9], a swarm intelligence algorithm [18] based on max–min ant system [19], Memetic Algorithm with 2-OPT and Bit-Flip search [17], and Joint 2-OPT and Bit-Flip [20], which changes the picking status of just one item whenever a segment in the cyclic tour is reversed.

In hyper-heuristic based methods, genetic programming is used to generate or select low level heuristics for the TSP and/or KP components. An individual in each generation is a tree whose internal nodes are simple arithmetic operators, while the leaf nodes are the numerical parameters of a given TTP instance. In [21], a genetic programming based approach generates two packing heuristics for the KP component. An individual in each generation is a tree whose internal nodes are functions while the leaf nodes correspond to low level heuristics. In [23], genetic programming is used to learn how to select a sequence of low level heuristics to address both the TSP and KP components. In [22], an individual in each generation is a Bayesian network in which each node corresponds to a low level heuristic. In [23], an individual in each generation is a tree in which the internal nodes are functions while the leaf nodes correspond to low level heuristics.
3. Background

A TTP instance has a set \{1, \ldots, n\} of cities and a set \{1, \ldots, m\} of items. The distance between each pair of cities \(i \neq i'\) is \(d(i, i') = d(i', i)\). Each item \(j\) is located at city \(i_j \geq 1\) (i.e., there are no items in the city 1). Furthermore, each item has weight \(w_j > 0\), profit \(\pi_j > 0\) and associated profitability ratio \(r_j = \pi_j/w_j\). An item \(j\) is considered more profitable than item \(j'\) if \(r_j > r_{j'}\), or \(\pi_j > \pi_{j'}\) if \(r_j = r_{j'}\).

The thief starts a cyclic tour from city 1, visits each city once and picks a subset of the items available in each city, and finally returns to city 1. The cyclic tour is represented by using a permutation of \(n\) cities. Given a cyclic tour \(c\), let \(c_k = i\) denote that the \(k\)-th city in the cyclic tour \(c\) is \(i\), and \(c(i) = k\) denote that the position of city \(i\) in the cyclic tour \(c\) is \(k\); as such, \(c_1 = 1\) and \(c(1) = 1\). A knapsack with a weight capacity \(W\) and a rent rate \(R\) per unit time is rented by the thief to carry the picked items. A picking plan \(p\) determines that item \(j\) is picked if \(p_j = 1\), or not picked if \(p_j = 0\).

A solution that comprises a cyclic tour \(c\) and a picking plan \(p\) is denoted as \((c, p)\).

The total weight of the items picked from city \(i\) is given by \(W_c(i) = \sum_{j=1}^{n} w_j p_j\). The total weight of the items picked from the first \(k\) cities in the cyclic tour \(c\) is given by \(W_{c,p}(k) = \sum_{i=1}^{k} W_c(i)\). The thief travels from city \(c_k\) to the next city with speed \(v_{c,p}(k)\) that decreases as \(W_{c,p}(k)\) increases. The speed at the city \(c_k\) is given by \(v_{c,p}(k) = v_{\text{max}} - W_{c,p}(k) \times (v_{\text{max}} - v_{\text{min}})/W\), where \(v_{\text{max}}\) and \(v_{\text{min}}\) are the given maximum and minimum speeds, respectively.

Given a cyclic tour \(c\) and a picking plan \(p\), the total profit is \(P(p) = \sum_{i=1}^{n} p_i \pi_i\). The travelling time to city \(c_k\) is \(T_{c,p}(k) = \sum_{i=1}^{k-1} d(c_i,c_{i+1})/v_{c,p}(k')\), and the total travelling time is \(T(c,p) = T_{c,p}(n+1) = T_{c,p}(n) + d(c_n,c_1)/v_{c,p}(n)\).

The goal of TTP is to maximise the objective function \(G(c,p) = P(p) - R \times T(c,p)\) over any possible \(c\) and \(p\). In other words, the goal is to maximise the total gain by maximising the total profit while at the same time minimising the total renting cost of the knapsack.

In a similar manner to the co-operational co-evolution approach [16], where a problem is divided to several sub-problems and each sub-problem is solved by a separate module, TTP is often decomposed to its TSP and KP components [15], with each component solved by a dedicated solver. In solving the TSP component, the picking plan \(p\) and hence \(W_c(i)\) for all cities \(1 \leq i \leq n\) are considered fixed; the aim is to minimise the total travelling time \(T(p,c)\) over any possible cyclic tour \(c\). In solving the KP component, the cyclic tour \(c\) and hence \(c_k\) for all positions \(1 \leq k \leq n\) are considered fixed; the aim is to maximise \(G(c,p)\) over any possible picking plan \(p\). To generate an initial solution, either a cyclic tour \(c\) is generated by assuming an empty picking plan \(p\) [12] (where no item is considered picked), or a picking plan \(p\) is generated assuming that all distances between cities are zero at the start [15].

For solving the TSP component, a segment reversing heuristic known as 2-OPT [10] is often used for modifying the cyclic tour \(c\). The underlying 2-OPT\((c,k',k'')\) function is defined as follows. Given a cyclic tour \(c\) with two positions \(k'\) and \(k''\) such that \(1 < k' < k'' \leq n\), the order of the visited cities in between the two positions is reversed to obtain a cyclic tour \(c'\). So \(c'_{k-k'} = c_{k-k'}\) is obtained where \(0 \leq k \leq k'' - k'\).

For solving the KP component, the bit-flip operator is often used for changing the picking plan \(p\). We define the flipping function as \(\text{Flip}(p,j)\), where given a picking plan \(p\) and a selected item \(j\), the picking state \(p_j\) is flipped from 0 to 1 or vice versa to obtain a new picking plan \(p'\).

To evaluate the effects of each application of 2-OPT\((c,k',k'')\) and \(\text{Flip}(p,j)\), the corresponding objective functions \(G(c',p)\) and \(G(c',p')\) must be recalculated. This necessitates recalculating \(W_{c',p}(k)\) and \(T_{c',p}(k+1)\) for all positions \(k' \leq k \leq k''\) in 2-OPT\((c,k',k'')\), as well as \(W_{c',p'}(k)\) and \(T_{c',p'}(k+1)\) for all positions \(l_j \leq k \leq n\) in \(\text{Flip}(p,j)\). This results in an overall computation cost of \(O(n)\) for \(G(c',p)\) and \(G(c',p')\).

In the following sections, we use two functions on sequences of numbers called prefix-minimum and postfix-maximum.

For any position \(k\) of a sequence of \(n\) numbers \(S = (S(1), S(2), \ldots, S(n))\), the prefix-minimum function is defined as \(\Pi(S,k) = \min(\Pi(S,k-1), S(k))\), where \(\Pi(S,1) = S(1)\), and the postfix-maximum function is defined as \(\Omega(S,k) = \max(\Omega(S,k-1), S(k+1))\), where \(\Omega(S,1) = S(n)\). In other words, the prefix-minimum function returns the smallest number among the first \(k\) numbers, while the postfix-maximum function returns the largest number among the last \(n-k+1\) numbers for each position \(k\) in the sequence of numbers \(S\). For example, consider the sequence \(S = (9, 6, 8, 4, 5, 7)\). The corresponding sequences generated via the prefix-minimum and postfix-maximum functions are \(\Pi(S) = (9, 6, 6, 4, 4, 4)\) and \(\Omega(S) = (9, 8, 8, 7, 7, 7)\), respectively.
4. Profit Guided Coordination Heuristic

In the definition of TTP given in Section 3, an item picking plan is required. The items are dispersed over the cities and their picking order is restricted by the order of the cities in the cyclic tour, which suggests that no monotonous item picking ordering should be expected in TTP. Constructive methods such as Insertion [9] and PackIterative [11] combine profitability ratios of the items with the distances of the respective cities from the end of the tour. However, in iterative methods which change the order of the cities in solving the TSP component, corresponding changes are required to the picking plan.

As mentioned in Section 3, the 2-OPT segment reversing heuristic is often used for solving the TSP component. As the length of the segment to be reversed increases, the amount of corresponding changes required for the picking plan is likely to increase. Within the context of a meta-optimiser that interleaves solving the TSP and KP components, the changes to the picking plan are postponed until the dedicated KP solver is executed. However, if a reversed segment is not accepted while solving the TSP component, there is no opportunity to evaluate corresponding changes to the picking plan for the KP component. This unnecessarily restricts the search space, as potentially beneficial combinations of segment reversal with corresponding changes to the picking plan are not even attempted.

As an example, consider the simple TTP instance shown in Figure 1, which has \( n = 5 \) cities and \( m = 4 \) items. Suppose that the capacity of knapsack \( W = 6 \), maximum speed \( v_{\text{max}} = 1 \), minimum speed \( v_{\text{min}} = 0.1 \) and the renting rate of the knapsack \( R = 1 \). Furthermore, suppose that an interim solution has the cyclic tour \( c = [1, 2, 3, 4, 5] \) and the picking plan \( p = [0, 0, 1, 1] \) (i.e., items 3 and 4 are picked). The objective value for this solution is \( G(c, p) = 4 \).

Using the 2-OPT heuristic for reversing the segment \([2, 3, 4]\) in the cyclic tour \( c \), we obtain the candidate cyclic tour \( c' = [1, 4, 3, 2, 5] \). Without changing the picking plan \( p \), the corresponding objective value is \( G(c', p) = -1.5 \), which can result in the rejection of the tour modification. However, if the picking plan is fortuitously changed to \( p' = [1, 0, 0, 1] \), where item 1 is picked and item 3 is unpicked, the resultant objective value is \( G(c', p') = 6 \). Hence changing the picking plan in coordination with reversing the segment can result in a higher total gain.

To see how to change the picking plan in coordination with segment reversing, let us first observe the solutions found by the PackIterative method (the main building block of the state-of-the-art S5 solver [11]). For a given solution \((c, p)\), we examine the least profitable item \( p(k) \) picked at each city \( c_k \) and the highest profitable item \( q(k) \) not picked at city \( c_k \) and plot the corresponding sequences of profitability ratios \( P_{c,p}(k) = r_{p(k)} \) and \( Q_{c,p}(k) = r_{q(k)} \), respectively. If no item is picked at a given city \( c_k \), we use \( P_{c,p}(k) = 1 + \max_i r_i \), where \( \max_i r_i \) is the maximum rate among all the items, as the default maximum value. Furthermore, if there are no unpicked items in city \( c_k \), we use \( Q_{c,p}(k) = 0 \), as the default minimum value.

Figure 2(a) shows the lowest picked and the highest unpicked profitability ratios for the benchmark instance “eil76_n750_uncorr_10.ttp” (see Section 7 for details on benchmark instances). Looking forwards (from the start to the end of the cyclic tour), the overall trend is a decrease in the lowest picked profitability ratios in the \( P_{c,p} \) sequence. Furthermore, looking backwards (from the end to the start of the cyclic tour), the overall trend is an increase in the highest unpicked profitability ratios in the \( Q_{c,p} \) sequence.

To capture the declining trend of the lowest picked profitability ratios in the \( P_{c,p} \) sequence looking forwards, we use the prefix-minimum function \( \Pi(P_{c,p}, k) \) (defined in Section 3) that returns the profitability ratio of the least profitable item picked in the first \( k \) cities. Furthermore, to capture the rising trend of the highest unpicked profitability ratios in the \( Q_{c,p} \) sequence looking backwards, we use the postfix-maximum function \( \Omega(Q_{c,p}, k) \) (also defined in Section 3) that

![Figure 1: An example TTP instance with 5 cities and 4 items.](image-url)
returns the profitability ratio of the highest profitable item not picked in the last \( n - k + 1 \) cities. Figure 2(a) shows the prefix-minimum and postfix-maximum values corresponding to the lowest picked and the highest unpicked profitability ratios, respectively.

Suppose that the 2-OPT move is applied on a segment between positions 39 and 74 of the cyclic tour shown in Figure 2(a). The resultant tour is shown in Figure 2(b), where the cities between the two positions are reversed. This also results in lower prefix-minimum values and higher postfix-maximum values than the corresponding original prefix-minimum and postfix-maximum values at most positions in the segment. In contrast to the original segment and the whole cyclic tour, the overall trend of the lowest picked profitability ratios in the reversed segment is rising (looking forwards), and the overall trend of the highest unpicked profitability ratios in the reversed segment is declining (looking backwards). This reversal in trends makes such a 2-OPT move counterproductive and results in a smaller total gain, which in turn may cause the move to be rejected by a solver. As the picking plan is not changed, the trade-off between profit and renting cost is poor for the low profitable items picked from the start of the reversed segment.

We propose to minimise the renting cost of the knapsack and maximise the profit of the picked items by adjusting the picking plan in coordination with reversal of the segment. To accomplish this, we propose an extended and modified form of the 2-OPT heuristic, denoted as Profit Guided Coordination Heuristic (PGCH). In addition to reversing the tour segment, items deemed as less profitable and picked in cities earlier in the reversed segment are first unpicked. The original prefix-minimum values at the given tour positions are used as the reference to decide which items must be unpicked. Then, items that tend to be equally or more profitable and not picked in cities later in the reversed segment are picked. The original postfix-maximum values at the given tour positions are used as the reference to decide which items can be picked.

For example, Figure 3(a) shows the lowest picked and the highest unpicked profitability ratios after reversing the segment from Figure 2(b), with an overlay of original prefix-minimum values from Figure 2(a). The highlighted green regions show items which must be unpicked from early positions of the reversed segment. The result of unpicking the required items is shown in Figure 3(b). Figure 4(a) shows the lowest picked and highest unpicked profitability ratios from Figure 3(b), with an overlay of original postfix-maximum values from Figure 2(a). Highlighted blue regions show items which can be picked from later positions of the reversed segment. The result of picking most of the items is shown in Figure 4(b). Figure 5 contrasts the effects of the 2-OPT and PGCH moves applied to the segment between positions 39 and 74 in Figure 2(a). For the modified tour, PGCH obtains a larger total gain than the original tour and 2-OPT.

We formally define PGCH\((c, p, k', k'')\) used in the previous example as follows. Given a TTP solution \((c, p)\) as well as positions \(k'\) and \(k''\) (under the condition \(1 < k' < k'' \leq n\)), a candidate solution denoted as \(\langle c', p' \rangle\) is obtained. Initially, \(p'\) is set to \(p\) and the cyclic tour \(c'\) is obtained such that \(c'_{j+k} = c_{k''-k} \) for \(0 \leq k \leq k'' - k'\). Then, for each \(k' \leq k \leq k''\), each item \(j : p'_j = 1\) from city \(c'_k = l_j\) is unpicked (ie. \(p'_j\) is set to 0) if \(r_j < \Pi(P_{c,p}, k)\). Furthermore, for each \(k'' \geq k \geq k'\), each item \(j : p'_j = 0\) from city \(c'_k = l_j\) is picked (ie. \(p'_j\) is set to 1) if \(r_j > \Omega(Q_{c,p}, k)\), provided that the total weight of the newly picked items is not larger than the total weight of the newly unpicked items in the reversed segment. Note that if no items are unpicked and replaced by other items, PGCH acts like a typical 2-OPT heuristic.

To evaluate the effects of each application of PGCH, data required by the objective function needs to be updated. Updating \(W_{c', p'}(c'_k)\) for all positions \(k' \leq k \leq k''\) as well as \(W_{c', p'}(k)\) and \(T_{c', p'}(k+1)\) for all positions \(k' \leq k \leq n\) results in the cost of \(O(\max(n, m))\) to compute the total gain.

---

Figure 2: x-axis: position in a cyclic tour; y-axis: profitability ratio. (a): Lowest picked and highest unpicked profitability ratios with the corresponding prefix-minimum and post-fix-maximum values in a solution found by the PackIterative method for the eil76_p750_uncorr_j0.tsp instance, where the total gain is 77544.88. (b): Effect of applying the 2-OPT move, where the segment between positions 39 and 74 is reversed and the picking plan is remained unchanged, resulting in a smaller total gain of 72151.46.
Figure 3: x-axis: position in a cyclic tour; y-axis: profitability ratio. (a): Lowest picked and highest unpicked profitability ratios of the reversed segment from Figure 2(b), with an overlay of original prefix-minimum values (dashed red line) from Figure 2(a). Highlighted green regions under the dashed red line and above the solid blue line indicate the items that must be unpicked. (b): Lowest picked and highest unpicked profitability ratios after unpicking the items as indicated in (a).

Figure 4: x-axis: position in a cyclic tour; y-axis: profitability ratio. (a): Lowest picked and highest unpicked profitability ratios from Figure 3(b), with an overlay of original postfix-maximum values (dashed yellow line) from Figure 2(a). Highlighted blue regions under the dotted green line and above the dotted-dashed yellow line indicate the items which can be picked. (b): Lowest picked and highest unpicked profitability ratios after picking most of the items as indicated in (a).

Figure 5: x-axis: position in a cyclic tour; y-axis: profitability ratio. (a): Copy of Figure 2(b), where the 2-OPT move is applied on the segment between positions 39 and 74 of Figure 2(a), resulting in a total gain of 72151.46. (b): Effect of applying the PGCH move instead of the 2-OPT move, resulting in an improved total gain of 78525.18.
5. Boundary Bit-Flip Search

A hill-climber known as bit-flip search has been previously used for solving the KP component within the TTP setting [6, 11]. The picking status is flipped (from picked to unpicked and vice-versa) of one item at a time. A change to the picking plan is kept if it improves the objective value. While this is straightforward, the downside is that each change to the picking plan is small and untargeted, resulting in a slow and meandering exploration of the solution space. When a time limit is placed for finding a solution, the solution space may not be explored adequately, which can contribute to poor quality TTP solutions.

We propose a more targeted form of bit-flip search, where a restriction is placed to consider changes only to the picking state of boundary items, which are defined as items whose profitability ratios have notable effect on prefix-minimum and post-fix-maximum values. Our motivation for this restriction is twofold: (i) changing the picking status of an item requires re-computation of the time to travel from the location of the item to the end of the cyclic tour, which can be a time consuming process, and (ii) given a fixed cyclic tour, high quality solutions usually follow the same pattern, where the lowest picked profitability ratios tend to decline when looking forwards, while the highest unpicked profitability ratios tend to rise up when looking backwards, as described in Section 4.

We formally define the boundary items as follows. Given a solution \((c, p)\), the least profitable item \(p(k)\) picked in city \(c_k\) is considered as a boundary item if it is the least profitable item among all the items picked at the first \(k\) cities, ie., \(\Pi(P_{c,p}, k) = P_{c,p}(k) = r_p(k)\). Furthermore, the highest profitable item \(q(k)\) unpicked in city \(c_k\) is considered as a boundary item if it is the highest profitable item among all the items unpicked at the last \(n-k+1\) cities, ie., \(\Omega(Q_{c,p}, k) = Q_{c,p}(k) = r_q(k)\). Considering any position \(k\), both the least profitable item among all the items picked at the first \(k\) cities and the highest profitable item among all the items unpicked at the last \(n-k+1\) cities are assumed as good candidates to be unpicked and picked, respectively. When the picking status of a boundary item located in position \(k\) is flipped, the values of \(\Pi(P_{c,p}, k')\) with \(k \leq k' \leq n\) and the values of \(\Omega(Q_{c,p}, k'')\) with \(1 < k'' \leq k\) must be updated. In both cases, the corresponding set of boundary items needs to be updated as well. Keeping up to date the position of city 1 is fixed as the first position in the tour.

### Cooperative Coordination Solver

The proposed PGCH and Boundary Bit-flip approaches are employed in the Cooperative Coordination (CoCo) solver shown in Algorithm 1. In the CoCoSolver() function, an initial cyclic tour is found using the well-known Chained Lin-Kernighan heuristic [14] via the ChainedLKTour() function. The InitPickingPlan() function provides an initial picking plan, which is the best plan (obtaining the largest TTP total gain) out of the plans provided by the Insertion [9] and Packiterative [11] methods. The initial solution is iteratively refined through the TSPSolver() and KPSolver() functions in an interleaved form. If in any iteration the solution provided by TSPSolver() is not improved by KPSolver() in terms of poor quality TTP solutions.

We propose a more targeted form of bit-flip search, where a restriction is placed to consider changes only to the picking state of boundary items, which are defined as items whose profitability ratios have notable effect on prefix-minimum and post-fix-maximum values. Our motivation for this restriction is twofold: (i) changing the picking status of an item requires re-computation of the time to travel from the location of the item to the end of the cyclic tour, which can be a time consuming process, and (ii) given a fixed cyclic tour, high quality solutions usually follow the same pattern, where the lowest picked profitability ratios tend to decline when looking forwards, while the highest unpicked profitability ratios tend to rise up when looking backwards, as described in Section 4.

We formally define the boundary items as follows. Given a solution \((c, p)\), the least profitable item \(p(k)\) picked in city \(c_k\) is considered as a boundary item if it is the least profitable item among all the items picked at the first \(k\) cities, ie., \(\Pi(P_{c,p}, k) = P_{c,p}(k) = r_p(k)\). Furthermore, the highest profitable item \(q(k)\) unpicked in city \(c_k\) is considered as a boundary item if it is the highest profitable item among all the items unpicked at the last \(n-k+1\) cities, ie., \(\Omega(Q_{c,p}, k) = Q_{c,p}(k) = r_q(k)\). Considering any position \(k\), both the least profitable item among all the items picked at the first \(k\) cities and the highest profitable item among all the items unpicked at the last \(n-k+1\) cities are assumed as good candidates to be unpicked and picked, respectively. When the picking status of a boundary item located in position \(k\) is flipped, the values of \(\Pi(P_{c,p}, k')\) with \(k \leq k' \leq n\) and the values of \(\Omega(Q_{c,p}, k'')\) with \(1 < k'' \leq k\) must be updated. In both cases, the corresponding set of boundary items needs to be updated as well. Keeping up to date the position of city 1 is fixed as the first position in the tour.

### Cooperative Coordination (CoCo) solver using the proposed Profit Guided Coordination Heuristic (PGCH) and Boundary Bit-Flip approaches

In TSPSolver(), \(n\) is the number of cities in the cyclic tour, while \(k'\) and \(k''\) indicate the starting and ending point, respectively, of the segment to be reversed. Each segment must have at least two cities. The position of city 1 is fixed as the first position in the tour.

#### Algorithm 1 Cooperative Coordination (CoCo) solver using the proposed Profit Guided Coordination Heuristic (PGCH) and Boundary Bit-Flip approaches

`proc CoCoSolver()`

\[
\langle c_*, p_* \rangle \leftarrow \emptyset \{ \text{best solution} \} \\
\text{while not global-timeout do} \\
\quad c \leftarrow \text{ChainedLKTour()} \\
\quad p \leftarrow \text{InitPickingPlan}(c) \\
\quad \text{repeat} \\
\quad \quad G_{\text{GSP}} \leftarrow G(c, p) \\
\quad \quad p \leftarrow \text{KPSolver}(c, p) \\
\quad \quad \text{if } G(c, p) = G_{\text{GSP}} \\
\quad \quad \quad \text{break} \{ \text{escape inner loop} \} \\
\quad \quad \text{end if} \\
\quad \quad \text{end while} \\
\quad \text{if } G(c, p) > G(c_*, p_*) \\
\quad \quad \langle c_*, p_* \rangle \leftarrow \langle c, p \rangle \\
\quad \text{end if} \\
\text{return } \langle c_*, p_* \rangle
\]

`proc TSPSolver(c, p)`

\[
\langle c_1, p_1 \rangle \leftarrow (c, p) \{ \text{best candidate solution} \} \\
\text{repeat} \\
\quad G_{\text{prev}} \leftarrow G(c, p) \\
\quad \text{for } k' \leftarrow 2 \text{ to } n - 1 \text{ do} \\
\quad \quad \text{foreach } (c_k, p_k) \in \text{DelTransNeigh}[c_k] \text{ with } k' < k' \leq n \\
\quad \quad \quad (c', p') \leftarrow \text{PGCH}(c, p, k', k'') \\
\quad \quad \quad \text{if } G(c', p') > G(c_1, p_1) \text{ then} \\
\quad \quad \quad \quad \langle c_1, p_1 \rangle \leftarrow (c', p') \{ \text{new best candidate solution} \} \\
\quad \quad \quad \text{end if} \\
\quad \quad \text{end foreach} \\
\quad \text{end for} \\
\quad \langle c, p \rangle \leftarrow \langle c_1, p_1 \rangle \{ \text{only the best PGCH move takes effect} \} \\
\quad \text{while } G(c, p) - G_{\text{prev}} \geq \alpha \cdot G_{\text{prev}} \{ \alpha = 0.1 \times 10^{-3} \} \\
\text{return } \langle c, p \rangle
\]

`proc KPSolver(c, p)`

\[
b \leftarrow \text{BoundaryItems}(c, p) \\
\text{MarkUnCheckedAll}(b) \\
\text{while not AllChecked(b) do} \\
\quad j \leftarrow \text{RandUnCheckedItem}(b) \\
\quad \text{MarkChecked}(j) \\
\quad p' \leftarrow \text{Flip}(p, j) \\
\quad \text{if } G(c, p') > G(c, p) \text{ then} \\
\quad \quad p \leftarrow p' \\
\quad \text{Update}(b) \\
\quad \text{MarkUnCheckedAll}(b) \\
\text{end if} \\
\text{end while} \\
\text{return } p
\]
the total gain value, the refining process of the current solution is aborted. The entire process is iteratively restarted until a global time limit is reached.

The TSPSolver() function is a steepest ascent hill-climbing method which behaves as follows. Given a TTP solution \(\langle c, p \rangle\), the best candidate solution \(\langle c_1, p_1 \rangle\) is first considered to be the same as \(\langle c, p \rangle\). Then, for each position \(k\) with \(1 < k < n\), the pre-computed Delaunay triangulation [24] neighbourhood for city \(c_k\) is considered, as done by other TSP solving algorithms [12]. For each city \(c_{k''} : k' < k'' \leq n\) in the neighbourhood specified by DelaTriNeighb\([c_k]\), the proposed PGCH move is applied to obtain a new candidate solution \(\langle c', p' \rangle\). If the total gain of the new candidate solution is larger than the best candidate solution found so far, it is accepted as the new best candidate solution. After checking all positions in the tour, the current solution \(\langle c, p \rangle\) is replaced by the best candidate solution \(\langle c_1, p_1 \rangle\). As such, only the best PGCH move takes effect and changes both \(c\) and \(p\). If the total gain of the current solution has sufficiently changed (with a margin empirically quantified as \(\alpha = 0.01\%\) of the previous solution), the check of all positions is repeated with the new current solution. Otherwise, the current solution is returned.

The value of the \(\alpha\) parameter has been empirically set with the aim to avoid spending time on improvements that are likely to be very minor, especially when solving large instances. Within a given time budget, this approach allows TSPSolver() to evaluate a larger number of tours, some possibly more promising.

In the KPSolver() function, all boundary items are placed in bag \(b\) via the BoundaryItems\((c, p)\) function, and are then marked as unchecked by the MarkUnCheckedAll\((b)\) function. The AllChecked\((b)\) function is used to determine whether all items in bag \(b\) have been checked. As long as there is at least one unchecked boundary item in the bag, a randomly unchecked boundary item \(j\) is selected via the RandUnCheckedItem\((b)\) function. The selected item is marked as checked via the MarkChecked\((j)\) function, and its picking status is flipped (from picked to unpicked or vice versa) to obtain a new candidate picking plan \(p'\). If the new total gain is larger than the current one, the change is accepted. In such case, the boundary items bag \(b\) is updated via the Update\((b)\) function, and all the items inside the bag are marked as unchecked via the MarkUnCheckedAll\((b)\) function. The initial filling of the items bag \(b\) is done as explained in Section 5.

7. Experiments

Three sets of experiments were performed on a broad subset of benchmark instances\(^2\) introduced in [6] and placed into 3 categories as per [12]. We denote the 3 categories as A, B, and C. Each category has 20 instances with a range of 76 to 33810 cities. In category A, there is only one item in each city; the profits and weights of the items are strongly correlated; knapsack capacity is relatively small. In category B, there are 5 items in each city; the profits and weights of the items are uncorrelated; the weights of the items are similar to each other; knapsack capacity is moderate. In category C, there are 10 items in each city; the profits and weights of the items are uncorrelated; knapsack capacity is high.

The same experiment setup was used in all experiments. All solvers were independently run on each TTP instance 10 times. Each run had a standard 10-minute timeout. For each run in all experiments, we ensured that each solver computes new initial cyclic tours via the Chained Lin-Kernighan heuristic [14] whenever required. All experiments were run on a machine with a 2 GB memory limit and an Intel Xeon CPU X5650 running at 2.66 GHz.

To measure differences in performance, for each solver on each TTP instance we use the relative deviation index [25], defined as:

\[
\text{RDI} = \left( \frac{G_{\text{mean}} - G_{\text{min}}}{G_{\text{max}} - G_{\text{min}}} \right) \times 100
\]

where \(G_{\text{mean}}\) is the mean of the \(G(c, p)\) total gain values of the 10 runs of the solver on an instance, while \(G_{\text{min}}\) and \(G_{\text{max}}\) are the minimum and the maximum \(G(c, p)\) values, respectively, obtained by any run of any solver on the same instance in each set of experiments. For each set of experiments, Appendix A contains corresponding tables that show the minimum and maximum \(G(c, p)\) values as well as the \(G_{\text{mean}}\) value for each solver on each instance.

In the first set of experiments we gauge the effects of the proposed PGCH move and Boundary Bit-Flip search, as described in Sections 4 and 5, respectively. We compare four variants of the TTP solver described in Section 6, denoted as Solver 1, Solver 2, Solver 3, and Solver 4. For solving the TSP component, either the standard 2-OPT move or the proposed PGCH move is used. For solving the KP component, either standard bit-flip search or the proposed Boundary Bit-Flip search is used. The variants are configured as follows:

- **Solver 1**: 2-OPT + standard bit-flip search
- **Solver 2**: 2-OPT + Boundary Bit-Flip search
- **Solver 3**: PGCH + standard bit-flip search
- **Solver 4**: PGCH + Boundary Bit-Flip search (equivalent to the CoCo solver described in Section 6)

\(^2\)The benchmark instances were obtained from https://cs.adelaide.edu.au/~optlog/CEC2014COMP_InstancesNew/
In the second set of experiments, we analyse the differences in performance between 2-OPT and PGCH in more detail. We compare Solver 2 and Solver 4 (the two best performing variants using 2-OPT and PGCH), based on the average relative length and the average number of the accepted segment reversing moves. The results shown in Table 2 indicate that on average PGCH leads to notably more accepted moves. Furthermore, accepted PGCH moves are considerably longer (i.e., larger segments). This supports our hypothesis in Section 4: it is better to change the picking plan in direct coordination with segment reversal, instead of postponing the changes and completely relying on the dedicated KP solver (i.e., weak coordination).
Table 2: Comparing 2-OPT move with the proposed PGCH move in Solver 2 and Solver 4, respectively. The first two columns in each category show the average relative lengths of the accepted segment reversing moves (in %), i.e., $|k'-k|+1 \times 100/n$. The last two columns in each category show the average number of accepted segment reversing moves.

| Instance | Rel. rev. len. % | Rev. num | Rel. rev. len. % | Rev. num | Rel. rev. len. % | Rev. num | Rel. rev. len. % | Rev. num |
|----------|------------------|----------|------------------|----------|------------------|----------|------------------|----------|
|          | Category A       |          | Category B       |          | Category C       |          |
|          | Solver 2 | Solver 4 | Solver 2 | Solver 4 | Solver 2 | Solver 4 | Solver 2 | Solver 4 |
| eil76    | 49.9      | 52.2     | 2.7       | 54.1     | 0.6       | 1.2     | 5.8       | 26.5     |
| kroA100  | 47.0      | 63.0     | 3.2       | 86.0     | 0.0       | 0.1     | 2.5       | 18.3     |
| ch130    | 7.1       | 30.2     | 1.7       | 48.1     | 0.5       | 2.3     | 1.6       | 54.0     |
| u159     | 2.2       | 21.9     | 1.3       | 21.0     | 2.5       | 4.4     | 1.3       | 30.7     |
| a280     | 46.8      | 62.6     | 0.8       | 49.7     | 1.2       | 5.6     | 0.9       | 33.8     |
| u574     | 15.2      | 28.7     | 2.5       | 21.3     | 2.3       | 8.0     | 2.1       | 25.7     |
| u724     | 16.9      | 30.5     | 2.6       | 26.3     | 3.6       | 17.6    | 2.0       | 37.3     |
| dsj1000  | 0.0       | 0.0      | 5.6       | 28.6     | 1.3       | 15.6    | 15.3      | 26.5     |
| ril1304  | 4.8       | 27.4     | 2.0       | 14.3     | 2.6       | 17.6    | 8.3       | 21.2     |
| fl1577   | 8.8       | 25.0     | 1.9       | 22.0     | 15.0      | 31.9    | 6.6       | 23.7     |
| d2103    | 10.9      | 30.2     | 15.3      | 51.9     | 4.0       | 12.3    | 23.2      | 45.5     |
| pch3038  | 18.8      | 32.7     | 1.0       | 29.0     | 3.7       | 33.7    | 0.6       | 36.1     |
| fnl4461  | 12.8      | 33.7     | 0.5       | 20.8     | 5.3       | 51.4    | 0.5       | 25.4     |
| pla7397  | 13.6      | 17.9     | 17.9      | 17.9     | 113       | 104     | 20.2      | 19.1     |
| rl11849  | 6.5       | 32.9     | 2.2       | 25.8     | 27.5      | 114     | 3.1       | 29.4     |
| usa3509  | 16.8      | 17.1     | 13.1      | 12.3     | 214       | 293     | 20.4      | 15.1     |
| brd14051 | 13.8      | 24.4     | 7.9       | 17.4     | 115       | 259     | 13.0      | 19.5     |
| d15112   | 12.3      | 28.3     | 10.5      | 20.1     | 174       | 265     | 16.1      | 22.6     |
| d18512   | 13.0      | 23.2     | 6.6       | 18.0     | 121       | 289     | 12.4      | 18.7     |
| pla33812 | 16.3      | 25.2     | 6.9       | 27.8     | 181       | 229     | 9.1       | 29.4     |
| Average  | 16.2      | 30.3     | 5.3       | 30.7     | 49.3      | 87.7    | 8.2       | 27.9     |

Table 3: Performance comparison of the CoCo solver (Solver 4 in Table 1) against MATLS [9], S5 [11] and CS2SA* [12] solvers. Performance is reported in terms of the relative deviation index (RDI), expressed as percentage, on 3 categories of TTP instances as per Table 1. Corresponding average total gain values are shown in Tables A4, A5 and A6 in Appendix A. Statistically significant differences between CoCo and the next best solver are marked with a star (*).

| Instance | Category A | Category B | Category C |
|----------|------------|------------|------------|
|          | MATLS      | S5         | CoCo       |
|          |            | CS2SA*     |            |
|          |            |            | CoCo       |
|          |            |            |            |
| eil76    | 72.2       | 100        | 36.3       | 100        |
| kroA100  | 76.9       | 67.2       | 15.2       | 99.2       |
| ch130    | 49.2       | 86.8       | 12.7       | 90.0       |
| u159     | 61.5       | 80.0       | 72.7       | 90.6       |
| a280     | 70.1       | 92.0       | 71.3       | 99.4       |
| u574     | 65.9       | 90.7       | 81.5       | 98.6       |
| u724     | 41.5       | 76.0       | 52.8       | 97.1       |
| dsj1000  | 92.4       | 4.8        | 43.6       | 98.0       |
| ril1304  | 48.8       | 88.0       | 67.9       | 94.6       |
| fl1577   | 54.6       | 93.4       | 75.8       | 94.6       |
| d2103    | 1.0        | 85.7       | 32.3       | 92.7       |
| pch3038  | 41.4       | 91.4       | 46.3       | 97.4       |
| fnl4461  | 30.6       | 84.4       | 82.1       | 91.5       |
| pla7397  | 69.3       | 95.5       | 76.9       | 98.9       |
| rl11849  | 27.3       | 87.0       | 54.7       | 95.1       |
| usa3509  | 40.9       | 94.1       | 68.1       | 97.9       |
| brd14051 | 41.5       | 91.4       | 69.9       | 96.4       |
| d15112   | 2.9        | 75.7       | 19.4       | 95.8       |
| d18512   | 51.5       | 93.9       | 81.4       | 97.5       |
| pla33810 | 23.7       | 87.6       | 59.3       | 95.1       |
| Average  | 48.2       | 83.3       | 66.1       | 96.5       |

The average number of accepted segment reversing moves.
In the third set of experiments, we compare the proposed CoCo solver (Solver 4 in Table 1) against the following solvers: MATLS [9], S5 [11] and CS2SA* [12]. The MATLS and S5 solvers were selected based on their notable performance reported in [13], while CS2SA* was selected due to its recency. The source code for MATLS and CS2SA* solvers was obtained from the respective authors. Table 3 shows the performance of all solvers in terms of RDI values.

Using analysis of variance (ANOVA) on the corresponding total gain values, followed by t-tests with a confidence interval of 95% [26], the proposed CoCo solver obtained statistically significantly better results than the next best method in the vast majority of cases. Similarly, in each category, the statistical significance of the differences between the average RDI values obtained for the CoCo solver and the next best solver were confirmed using a paired t-test with a confidence interval of 95%. Each statistically significant difference is marked with a star in Table 3. Overall, these results indicate that the techniques used by the CoCo solver, especially the explicit coordination in solving the TSP and KP components, as provided by the proposed PGCH move, are indeed beneficial.

8. Conclusion

Many real-world constraint optimisation problems, such as supply chain problems, comprise two or more interdependent components. Compared to solving a single component, the interdependency makes finding good overall solutions considerably more challenging, as finding an optimal solution to each component separately does not guarantee finding an optimal overall solution to the whole problem.

In the travelling thief problem (TTP), a thief undertakes a cyclic tour through a set of cities, and according to a picking plan, picks a subset of available items into a rented knapsack with limited capacity. TTP can be thought of as a combination of two interdependent components: the travelling salesman problem (TSP) and the knapsack problem (KP). A TTP solution includes a cyclic tour over the cities as a solution to the TSP component, and a plan for picking a subset of the available items as a solution to the KP component. Inspired by the co-operative co-evolution approach [16], methods to solving TTPs often involve solving the TSP and KP components in an interleaved fashion via dedicated component solvers [15]. When the solution to one component is changed, the other solution remains fixed.

In the TTP setting, items are scattered over the cities, with the order of the cities in the tour restricting the order of picking the items. As such, changing the order of the cities in the tour requires corresponding changes to the picking plan. The 2-OPT segment reversing heuristic [10] is often used for solving the TSP component. As the length of the segment to be reversed increases, the amount of corresponding changes required for the picking plan is likely to increase. Within the context of a meta-optimiser that interleaves solving the TSP and KP components, changes to the picking plan are postponed until the dedicated KP solver is executed. However, if a reversed segment is not accepted while solving the TSP component due to obtaining a lower objective value, there is no opportunity to evaluate corresponding changes to the picking plan for the KP component. This unnecessarily restricts the search space, as potentially beneficial combinations of segment reversal with corresponding changes to the picking plan are not even attempted.

To address the above issue, we have proposed a new heuristic for solving the TSP component, termed Profit Guided Coordination Heuristic (PGCH). Whenever a segment in the cyclic tour is reversed, the picking plan is adjusted accordingly. Items deemed as less profitable and picked in cities earlier in the reversed segment are replaced by items that tend to be equally or more profitable and not picked in the later cities in the reversed segment. Using PGCH for solving the TSP component, segment reversing moves that are longer than 2-OPT tend to be accepted. As a result, the quality of the cyclic tour is considerably improved.

We have also proposed to further change the picking plan through a modified form of the hill-climbing bit-flip search, where changes in the picking state are only permitted for boundary items. Such items are defined as the lowest profitable picked items or highest profitable unpicked items. This restriction reduces the amount of time spent on solving the KP component, thereby allowing more tours to be evaluated by the TSP component within a given time budget.

The two modified heuristics form the basis of a new cooperative coordination (CoCo) solver. On a broad range of benchmark TTP instances, the proposed CoCo solver has been shown to outperform several notable solvers: MATLS [9], S5 [11] and CS2SA* [12].

Acknowledgements

This article is a revised and extended version of our earlier work published at the International Symposium on Combinatorial Search [1]. We would like to thank our colleagues at Data61/CSIRO (Toby Walsh, Phil Kilby, Regis Riveret) for discussions leading to the improvement of this article.
References

[1] M. Namazi, M. A. H. Newton, A. Sattar, C. Sanderson, A profit guided coordination heuristic for travelling thief problems, in: International Symposium on Combinatorial Search, AAAI, 2019, pp. 140–144.

[2] F. Rossi, P. Van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier, 2006.

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

[4] Z. Michalewicz, Quo vadis, evolutionary computation?, in: Advances in Computational Intelligence, Lecture Notes in Computer Science (LNCS), Vol. 7311, 2012, pp. 98–121.

[5] M. R. Bonyadi, Z. Michalewicz, M. Wagner, F. Neumann, Evolutionary computation for multicomponent problems: opportunities and future directions, in: Optimization in Industry, Springer, 2019, pp. 13–30.

[6] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, F. Neumann, A comprehensive benchmark set and heuristics for the traveling thief problem, in: Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 477–484.

[7] G. Gutin, A. P. Punnen, The Traveling Salesman Problem and Its Variations, Springer, 2006.

[8] H. Kellerer, U. Pferschy, D. Pisinger, Introduction to NP-completeness of knapsack problems, in: Knapsack Problems, Springer, 2004, pp. 483–493.

[9] Y. Mei, X. Li, X. Yao, Improving efficiency of heuristics for the large scale traveling thief problem, in: Simulated Evolution and Learning, Lecture Notes in Computer Science (LNCS), Vol. 8886, 2014, pp. 631–643.

[10] G. A. Croes, A method for solving traveling-salesman problems, Operations Research 6 (6) (1958) 791–812.

[11] H. Faulkner, S. Polyakovskiy, T. Schultz, M. Wagner, Approximate approaches to the traveling thief problem, in: Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 385–392.

[12] M. El Yafrani, B. Ahiod, Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing, Information Sciences 432 (2018) 231–244.

[13] M. Wagner, M. Lindauer, M. Misur, S. Nallaperuma, F. Hutter, A case study of algorithm selection for the traveling thief problem, Journal of Heuristics 24 (3) (2018) 295–320.

[14] D. Applegate, W. Cook, A. Rohe, Chained Lin-Kernighan for large traveling salesman problems, INFORMS Journal on Computing 15 (1) (2003) 82–92.

[15] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, A. Wierzbicki, Socially inspired algorithms for the travelling thief problem, in: Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 421–428.

[16] M. A. Potter, K. A. De Jong, A cooperative coevolutionary approach to function optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 1994, pp. 249–257.

[17] M. El Yafrani, B. Ahiod, Population-based vs. single-solution heuristics for the travelling thief problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2016, pp. 317–324.

[18] M. Wagner, Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem, in: Swarm Intelligence, Lecture Notes in Computer Science (LNCS), Vol. 9882, 2016, pp. 273–281.

[19] T. Stützle, H. H. Hoos, MAX-MIN ant system, Future Generation Computer Systems 16 (8) (2000) 889–914.

[20] M. El Yafrani, B. Ahiod, A local search based approach for solving the Travelling Thief Problem: The pros and cons, Applied Soft Computing 52 (2017) 795–804.

[21] Y. Mei, X. Li, F. Salim, X. Yao, Heuristic evolution with genetic programming for traveling thief problem, in: IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 2753–2760.
[22] M. S. Martins, M. El Yafrani, M. R. Delgado, M. Wagner, B. Ahiod, R. Lüders, HSEDA: A heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2017, pp. 361–368.

[23] M. El Yafrani, M. Martins, M. Wagner, B. Ahiod, M. Delgado, R. Lüders, A hyperheuristic approach based on low-level heuristics for the travelling thief problem, Genetic Programming and Evolvable Machines 19 (1-2) (2018) 121–150.

[24] B. Delaunay, Sur la sphère vide, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk 7 (1934) 793–800.

[25] J.-U. Kim, Y.-D. Kim, Simulated annealing and genetic algorithms for scheduling products with multi-level product structure, Computers & Operations Research 23 (9) (1996) 857–868.

[26] R. Peck, T. Short, C. Olsen, Introduction to statistics and data analysis, Cengage Learning, 2019.
Appendix A. Total Gain Values

Tables A1 through to A6 provide the total gain values used for obtaining the RDI values in Section 7.

Table A1: Mean total gain values each over 10 runs corresponding to the solvers in Table 1 with the minimum and maximum total gain values among all 40 runs on Category A instances.

| Instance | Solver 1 | Solver 2 | Solver 3 | Solver 4 | Min      | Max      |
|----------|----------|----------|----------|----------|----------|----------|
| eil76    | 4109     | 4109     | 4109     | 4109     | 4109     | 4109     |
| kroA100  | 4740     | 4808     | 4855     | 4827     | 4601     | 4868     |
| ch130    | 9520     | 9506     | 9564     | 9564     | 9470     | 9564     |
| u159     | 8635     | 8634     | 8842     | 8820     | 8634     | 8979     |
| a280     | 18441    | 18347    | 18668    | 18636    | 18437    | 18741    |
| u574     | 27223    | 27140    | 27360    | 27367    | 26987    | 27517    |
| u724     | 50353    | 50425    | 51149    | 51124    | 50274    | 51522    |
| dsj1000  | 144219   | 144219   | 144219   | 144219   | 144219   | 144219   |
| ril1304  | 81283    | 81353    | 81611    | 81769    | 80862    | 82470    |
| b13777   | 93135    | 93006    | 93392    | 93030    | 91702    | 93934    |
| d2103    | 121533   | 121411   | 122175   | 121977   | 120874   | 122719   |

Table A2: Mean total gain values each over 10 runs corresponding to the solvers in Table 1 with the minimum and maximum total gain values among all 40 runs on Category B instances.

| Instance | Solver 1 | Solver 2 | Solver 3 | Solver 4 | Min      | Max      |
|----------|----------|----------|----------|----------|----------|----------|
| eil76    | 21647    | 21349    | 22269    | 22240    | 20854    | 22318    |
| kroA100  | 41330    | 41365    | 45503    | 45139    | 42471    | 45812    |
| ch130    | 61290    | 61290    | 61623    | 61702    | 61286    | 61703    |
| u159     | 60719    | 61016    | 60770    | 60987    | 60626    | 61077    |
| a280     | 110180   | 110302   | 116454   | 116457   | 110151   | 116458   |
| u574     | 25680    | 25739    | 260198   | 260062   | 256245   | 261036   |
| u724     | 307831   | 308087   | 321335   | 322133   | 306650   | 324316   |
| dsj1000  | 347065   | 347201   | 370065   | 370435   | 345482   | 373254   |
| ril1304  | 577423   | 585897   | 598058   | 598610   | 577335   | 599351   |
| b13777   | 615705   | 619338   | 630214   | 635696   | 599956   | 648171   |
| d2103    | 892467   | 896308   | 923100   | 925808   | 884237   | 929758   |
| pch3038  | 1180523  | 1182342  | 1198340  | 1204897  | 1173589  | 1207834  |
| fnl4461  | 594187   | 594346   | 594521   | 595687   | 578346   | 600571   |
| pla7397  | 430587   | 434070   | 4424384  | 441709   | 414150   | 465983   |
| ril1849  | 4576428  | 4633472  | 4723718  | 4780661  | 452879   | 481385   |
| usa13509 | 7801763  | 8161340  | 8118937  | 8254572  | 6848256  | 8340377  |
| brd14051 | 6428360  | 6615466  | 6740590  | 6816674  | 6090832  | 6870081  |
| d15112   | 705668   | 7393476  | 7470491  | 7714270  | 6576653  | 7909260  |
| d18512   | 6847516  | 7247641  | 7391911  | 7467890  | 6269585  | 7523963  |
| pla33810 | 15153466 | 15460892 | 16010375 | 16172664 | 14983123 | 16418169 |
Table A3: mean total gain values each over 10 runs corresponding to the solvers in Table 1 with the minimum and maximum total gain values among all 40 runs on Category C instances.

| Instance | Solver 1 | Solver 2 | Solver 3 | Solver 4 | Min    | Max    |
|----------|----------|----------|----------|----------|--------|--------|
| eil76    | 87476    | 87323    | 87270    | 87249    | 86366  | 88062  |
| kroA100  | 155977   | 155621   | 157735   | 157712   | 155585 | 158812 |
| ch130    | 207159   | 207081   | 207530   | 206851   | 206381 | 207902 |
| u159     | 246602   | 246493   | 248815   | 248627   | 246132 | 249312 |
| a280     | 429095   | 429095   | 429136   | 429138   | 429082 | 429138 |
| u574     | 967068   | 967355   | 969973   | 969666   | 966227 | 970692 |
| u724     | 1192048  | 1191552  | 1203543  | 1204366  | 1187571| 1205747|
| dsj1000  | 1484785  | 1487642  | 1490023  | 1495584  | 1480693| 1500167|
| r1304    | 2188104  | 2188936  | 2209944  | 2212260  | 2183617| 2214083|
| d1577    | 2462610  | 2479097  | 2482379  | 2491120  | 2448429| 2504641|
| d2103    | 3453655  | 3457755  | 3489555  | 3493856  | 3447868| 3503079|
| pch3038  | 4564948  | 4572248  | 4591221  | 4596155  | 4553969| 4607306|
| falc461  | 6539711  | 6533039  | 653496   | 6569715  | 6526013| 6573564|
| plat397  | 13873075 | 14393277 | 14258323 | 14496916 | 13328394| 14600106|
| r11849   | 18196554 | 18268196 | 18398981 | 18512214 | 1809364 | 18544745|
| usal3509 | 2546366  | 26391280 | 26122134 | 26572209 | 2428292 | 26743719|
| dsl4051  | 31805915 | 31469490 | 31428233 | 31418367 | 29142204| 314260278|
| d15112   | 26005329 | 27232945 | 27146384 | 27437349 | 25246146| 27695658|
| d18512   | 25478702 | 27373733 | 27439587 | 27690630 | 24056519| 27946341|
| pla33810 | 56305249 | 57695280 | 58151982 | 58744246 | 55320897| 58857245|

Table A4: Mean total gain values each over 10 runs corresponding to the solvers in Table 3 with the minimum and maximum total gain values among all 40 runs on Category A instances.

| Instance | MATLS | S5   | CS2SA* | CoCo   | Min    | Max    |
|----------|-------|------|--------|--------|--------|--------|
| eil76    | 3717  | 4109 | 3209   | 4109   | 2697   | 4109   |
| kroA100  | 4703  | 4650 | 4362   | 4827   | 4278   | 4831   |
| ch130    | 8868  | 9382 | 8779   | 9564   | 8193   | 9564   |
| u159     | 8314  | 8634 | 8105   | 8820   | 7252   | 8979   |
| a280     | 17539 | 18411| 16648  | 18636  | 15169  | 18692  |
| u574     | 25581 | 27007| 24043  | 27367  | 22866  | 27430  |
| u724     | 48865 | 50265| 47889  | 51124  | 47174  | 51244  |
| pch1000  | 143699| 137740| 144219 | 144219 | 137410 | 144219 |
| r1304    | 75804 | 80911| 75040  | 81769  | 69445  | 82470  |
| d1577    | 88330 | 93081| 83555  | 93030  | 81638  | 93895  |
| d2103    | 112686| 121274| 118997 | 121977 | 112584 | 122719 |
| pch3038  | 148995| 160115| 144999| 161456 | 139791 | 160230 |
| falc461  | 248482| 262478| 241881 | 264332 | 240500 | 266549 |
| plat397  | 367247| 395655| 333910 | 399349 | 291891 | 400596 |
| r11849   | 662904| 708215| 648843 | 714396 | 642243 | 718089 |
| usal3509 | 743146| 808583| 738638 | 813324 | 692847 | 815853 |
| brdl4051 | 813977| 875741| 786919 | 818914 | 762545 | 886332 |
| d15112   | 871348| 948696| 890746 | 967701 | 868378 | 972050 |
| d18512   | 996820| 1073310| 941041 | 1079883| 903478 | 108485 |
| pla33810 | 1730997| 1897560| 1736746| 1917177| 1669331| 1929990|
Table A5: Mean total gain values each over 10 runs corresponding to the solvers in Table 3 with the minimum and maximum total gain values among all 40 runs on Category B instances.

| Instance | MATLS  | S5   | CS2SA* | CoCo  | Min   | Max   |
|----------|--------|------|--------|-------|-------|-------|
| eil76    | 22286  | 22255| 17484  | 22240 | 13932 | 22626 |
| kroA100  | 43310  | 41192| 39109  | 45139 | 38865 | 45812 |
| ch130    | 60705  | 61071| 51572  | 61702 | 48856 | 61703 |
| u159     | 58778  | 60550| 55662  | 60897 | 44919 | 61077 |
| a280     | 111728 | 109932| 104863 | 116457| 99991 | 116457|
| a274     | 339630 | 345179| 324412 | 370435| 314896| 370435|

Table A6: Mean total gain values each over 10 runs corresponding to the solvers in Table 3 with the minimum and maximum total gain values among all 40 runs on Category C instances.

| Instance | MATLS  | S5   | CS2SA* | CoCo  | Min   | Max   |
|----------|--------|------|--------|-------|-------|-------|
| eil76    | 88131  | 88025| 84476  | 87249 | 76964 | 88332 |
| kroA100  | 155500 | 155582| 152621 | 157712| 149656| 158812|
| ch130    | 205552 | 206981| 206851 | 207881| 193430| 207881|
| u159     | 242452 | 246212| 241458 | 249138| 238934| 249138|
| a280     | 426951 | 429014| 415418 | 429138| 407221| 429138|
| a274     | 1187092| 1190001| 1163870| 1204366| 1149297| 1204366|
| d2103    | 164547 | 1178341| 1153843| 1207834| 1127401| 1207834|
| pch3038  | 7709076| 790496| 789622  | 8254572| 7604959| 8317858|
| rhl1849  | 458308 | 4620657| 470733 | 4780661| 4304699| 4813852|
| usa13509 | 77709076| 790496| 789622  | 8254572| 7604959| 8317858|
| bed14051 | 8647038 | 6535639| 6191198 | 6816674| 5528880| 6870081|
| d15112   | 6797737 | 7135673| 7251647 | 7714270| 6530450| 7909260|
| d18512   | 7102985 | 7215763| 6562282 | 7467890| 5296818| 7515204|
| pla33810 | 1560536 | 15615480| 14918919| 16172664| 14420806| 16418169|