ReinforceNS: Reinforcement Learning-based Multi-start Neighborhood Search for Solving the Traveling Thief Problem

Tao Wu\textsuperscript{1}, Huachao Cui\textsuperscript{2}, Tao Guan\textsuperscript{1}, Yuesong Wang\textsuperscript{1} and Yan Jin\textsuperscript{1}\textsuperscript{*}

\textsuperscript{1}Huazhong University of Science and Technology
\textsuperscript{2}Nanjing Research Institute of Electronic Engineering
m202273706@hust.edu.cn, jinyan@mail.hust.edu.cn

Abstract

The Traveling Thief Problem (TTP) is a challenging combinatorial optimization problem with broad practical applications. TTP combines two NP-hard problems: the Traveling Salesman Problem (TSP) and Knapsack Problem (KP). While a number of machine learning and deep learning based algorithms have been developed for TSP and KP, there is limited research dedicated to TTP. In this paper, we present the first reinforcement learning based multi-start neighborhood search algorithm, denoted by ReinforceNS, for solving TTP. To accelerate the search, we employ a pre-processing procedure for neighborhood reduction. A TSP routing and an iterated greedy packing are independently utilized to construct a high-quality initial solution, further improved by a reinforcement learning based neighborhood search. Additionally, a post-optimization procedure is devised for continued solution improvement. We conduct extensive experiments on 60 commonly used benchmark instances with 76 to 33810 cities in the literature. The experimental results demonstrate that our proposed ReinforceNS algorithm outperforms three state-of-the-art algorithms in terms of solution quality with the same time limit. In particular, ReinforceNS achieves 12 new results for 18 instances publicly reported in a recent TTP competition. We also perform an additional experiment to validate the effectiveness of the reinforcement learning strategy.

1 Introduction

The travelling Thief Problem (TTP) is a challenging combinatorial optimization problem as it combines two NP-hard problems: the Traveling Salesman Problem (TSP) \cite{Flood, Knapsack} and Knapsack Problem (KP) \cite{Pisinger}. And it holds practical importance with various applications in real-life scenarios, such as water tank transportation \cite{Bonyadi}, printed circuit board design \cite{Reinelt}, airport ground movement \cite{Weiszer}, wavelength assignment for mesh networks \cite{Hu} and the routing problem with loading constraints \cite{Iori, Martello, Wagner}. Given its theoretical challenge and practical significance, TTP has attracted increasing research effort in the last decade \cite{Wagner, ElYafrani, Namazi}. Existing research on TTP includes exact algorithms, heuristics and hyper-heuristics. Due to its intrinsic intractability, the research lies in the development of effective heuristics to find near-optimal solutions within a reasonable time limit. Compared to the close relative problems TSP and KP that have been intensively studied \cite{Iori, Lienland, Osaba, Cacchiani}, TTP has received limited attention. Moreover, the TSP and KP algorithms integrating machine learning techniques have demonstrated remarkable performances \cite{Afshar, Zheng, Pierotti, Xin, Tu, Pan}, while few efforts have been dedicated to developing machine learning-based TTP algorithms \cite{Namazi, Pan}.

To the best of our knowledge, no study has been reported in the literature investigating the interest of the reinforcement learning technique for solving TTP. In this work, we fill the gap by introducing the first reinforcement learning-based multi-start neighborhood search algorithm ReinforceNS for TTP. ReinforceNS integrates pre-processing, initialization, reinforcement learning based neighborhood search and post-optimization. To ensure effective exploration of the search space, ReinforceNS employs a multi-start mechanism to escape local optima traps and the dedicated initialization to regenerate new high-quality initial solutions. The main contributions of the paper are summarized as follows.

- The proposed ReinforceNS algorithm integrates a pre-processing procedure and a reinforcement learning-based neighborhood search procedure to ensure an effective and efficient examination of promising neighboring solutions.
- ReinforceNS applies a post-optimization procedure to further improve the local optimal solutions. This is ensured jointly by an item flipping phase and a route refining phase.
- ReinforceNS is extensively tested on the commonly used 60 TTP benchmark instances. Experimental results show the proposed algorithm outperforms three state-of-

\textsuperscript{*}Yan Jin is the corresponding author, jinyan@mail.hust.edu.cn.
the-art algorithms within the same time limit. In particular, it finds 12 improved results for 18 instances publicly reported in a recent TTP competition.

2 Related Work

The TTP problem was first introduced in 2013 [Bonyadi et al., 2013] and has received increasing attention in recent years. The existing approaches for solving TTP can be mainly classified into three families: exact algorithms, heuristics and hyper-heuristics. In this section, we briefly review exact algorithms and hyper-heuristics, and then focus on heuristics.

Exact algorithms utilizing dynamic programming or mixed-integer programming [Polyakovskiy and Neumann, 2017; Wu et al., 2017] can find optimal solutions with theoretical guarantees, but face challenges when dealing with large instances within reasonable time due to the inherent exponential complexity.

In contrast, hyper-heuristic and heuristic algorithms can find near-optimal solutions for practical applications, but they lack optimality guarantees. The hyper-heuristic algorithms often utilize genetic programming to generate or select low-level heuristics for exploring routes or packing plans, such as [Mei et al., 2015], [Martins et al., 2017; El Yafrani and Ahiod, 2018], [Martins et al., 2017], [El Yafrani and Ahiod, 2018], [Ali and Mohamedkhai, 2021].

The heuristics are widely studied for solving TTP, which can be classified into the following three families [Namazi et al., 2023].

Constructive paradigm. The Chained Lin-Kernighan Heuristic (CLKH) [Applegate et al., 2003] is used to generate a route and greedy packing strategies are designed while keeping this route unchanged, such as [Polyakovskiy et al., 2014; Bonyadi et al., 2014; Faulkner et al., 2015]. S5 applies them within its restart-based framework [Faulkner et al., 2015]. The constructive paradigm often serves as an initialization procedure for other TTP algorithms.

Cooperative paradigm. After the generation of an initial TTP solution through the constructive paradigm, two separate search procedures for TSP and KP are iteratively applied to improve the route and packing plans. Some notable algorithms in this family include the Cooperative Solver (CoSolver) [Bonyadi et al., 2014] and its improved versions: CoSolver with 2-opt and Simulated Annealing (CS2SA) [El Yafrani and Ahiod, 2016], CS2SA with offline instance-based parameter tuning (CS2SA*) [El Yafrani and Ahiod, 2018], CoSolver with Reverse-Weighted Selection (RWS) [Zhang et al., 2021], as well as a state-of-the-art heuristic Cooperation Coordination (CoCo) proposed most recently [Namazi et al., 2023].

Full-encoding paradigm. Full-encoding paradigm addresses TTP without segregating it into TSP and KP. Many algorithms adopt a hybrid evolutionary framework [El Yafrani and Ahiod, 2016; El Yafrani and Ahiod, 2017; Mei et al., 2014; Wujiets and Thierens, 2019], such as Memetic Algorithm with Two-Stage Local Search (MATLS) [Mei et al., 2014], Memetic algorithm with Edge Assembly and 2-Point crossover operator (MEA2P) [Wujiets and Thierens, 2019]. Additionally, a swarm intelligence algorithm MMAS [Wagner, 2016] based on the Max-Min Ant System [Stützle and Hoos, 2000] has been proposed for solving TTP.

Note that three best-performing algorithms, i.e., S5, CoCo and MATLS, from each family of heuristics are used for comparisons in the experiments.

3 Problem Description

The TTP problem is defined as follows. Let $G = (N, E)$ be a complete graph, where $N = \{1, \ldots, n\}$ is the vertex set with 1 being the starting-ending city (depot) $x_1$ and other vertices representing $n-1$ other cities and $E$ is the set of edges. Let $D = (d_{i,j})$ be a non-negative distance matrix associated with $E$. A set of items $M = \{1, \ldots, m\}$ is distributed among these cities except city $x_1$. Suppose that each city $x_i (i \in \{2, \ldots, n\})$ contains $m_i$ distinct items, denoted as $M_i = \{i_1, \ldots, i_{m_i}\}, M = \bigcup_{i=1}^{n} M_i$ and $M_i \cap M_j = \emptyset, i \neq j$. Each item $i_k (i_k \in \{1, \ldots, m\})$ positioned in the city $x_i$ has a profit value $p_{i_k}$ and weight $w_{i_k}$. The thief starts from city $x_1$, visits each of the cities exactly once, and returns back to the starting city $x_1$. During the tour, the thief can pick up any item into his knapsack in any city until the total weight of items does not exceed the maximum weight $W$ of the knapsack. A renting rate $R$ is to be paid per each time unit being on the tour.

The $v_{\text{min}}$ and $v_{\text{max}}$ are the minimum and maximum speeds that the thief can move respectively. Suppose that $v_{\text{x}}$, represents the speed of the thief in city $x_i$. Eq. (1) gives the linear relationship between the thief’s speed and the knapsack’s weight.

$$v_{\text{x}} = v_{\text{max}} - C \times w_{\text{x}_i} \tag{1}$$

where $C = \frac{v_{\text{max}} - v_{\text{min}}}{W}$ is a constant value, $w_{\text{x}_i}$ indicates the knapsack weight of the thief in city $x_i$, as shown in Eq. (2).

$$w_{\text{x}_i} = \sum_{j=1}^{m} w_{j_k} \times z_{j_k} \text{ s.t. } z_{j_k} \in \{0, 1\} \tag{2}$$

Let $\Pi = (x_1, \ldots, x_n)$ represent a route plan and $P = (z_1, \ldots, z_m)$ be a binary vector representing a packing plan, where $z_j$ equals to 1 when item $j$ is picked up, and 0 otherwise. The objective of TTP is to find a route plan $\Pi$ and a packing plan $P$ of the maximal profit value as Eq. (3).

$$f(\Pi, P) = g(P) - R \cdot \psi(\Pi, P) \tag{3}$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{i_k} z_{i_k} - R \cdot \left( \frac{d_{x_i, x_1}}{v_{\text{max}} - C \times w_{\text{x}_i}} + \sum_{i=1}^{n-1} \frac{d_{x_i, x_{i+1}}}{v_{\text{max}} - C \times w_{\text{x}_i}} \right)$$

where $g(P)$ represents the total profit values of all items packed in the knapsack, $R \cdot \psi(\Pi, P)$ represents the amount that the thief pays for the knapsack’s rent.}

One easily notices that TTP can be reduced to TSP by assigning zero to the profit values of all items. Similarly, TTP can be considered as KP by setting the renting value to zero. These observations also confirm the challenge of solving the TTP problem.
4 The Proposed ReinforceNS Algorithm

The proposed ReinforceNS algorithm is a multi-start iterated neighborhood search heuristic reinforced by a reinforcement learning mechanism. After pre-processing, it explores the search space by repeating a dedicated solution construction, local optimization and post-optimization. The local optimization and post-optimization procedures apply neighborhood search by transitioning from the current solution to one of its neighboring solutions. Each iteration involves replacing the current solution with the best neighboring solution that offers an immediate improvement over the current solution.

Algorithm 1 provides the general scheme of ReinforceNS that relies on a multi-start iterated local search framework. Specifically, after the initialization of the optimal solution \( (\Pi^*, P^*) \) and its corresponding objective value \( f^* \), ReinforceNS applies a pre-processing procedure to reduce the candidate edge sets, and then performs a series of iterations until reaching the maximum time limit \( T_{\text{max}} \). At each iteration, a solution \( \Pi \) is generated by the TSP solver CLKH [Applegate et al., 2003]. Subsequently, an iterated greedy packing procedure (IGP) is applied to \( \Pi \) to produce a packing plan \( P \), thus initializing a high-quality solution \( (\Pi, P) \) for TTP. This solution is improved through the reinforcement learning based neighborhood search procedure. To help the search escape from the local optima, a post-optimization procedure including an item flipping phase and a route refining phase is employed for further improvement. The best-found solution \( (\Pi^*, P^*) \) is updated and finally returned as the output at the end of the ReinforceNS algorithm.

4.1 Pre-Processing

Before starting the search, we pre-process the input TTP instance \( G \) using a neighborhood reduction procedure. This is achieved by a subgradient optimization approach on the 1-tree and \( \alpha \)-nearness adopted from the well-known TSP algorithm Lin-Kernighan-Heldsgaun (LKH) [Heldsgaun, 2000].

Given a complete graph \( G \), the neighborhood \( \mathcal{N}(x_i) \) of city \( x_i \) is defined as the set of adjacent cities \( x_j \) connected to \( x_i \) by an edge \( (x_i, x_j) \). Initially, each vertex has \( n-1 \) neighbors. A 1-tree of \( G \) is a spanning tree for the graph of vertex set \( V \setminus \{x_i\} \) combined with two edges in \( E \) connected to vertex \( x_i \), an arbitrary special vertex in \( V \). A minimum 1-tree \( T \) is the 1-tree with minimum length. The \( \alpha \)-nearness of an edge \( (x_i, x_j) \in E \) is defined as \( \alpha(i,j) = L(T^+(x_i, x_j)) - L(T) \), where \( L(T) \) is the length of \( T \) and \( L(T^+(x_i, x_j)) \) is the length of minimum 1-tree that includes the edge \( (x_i, x_j) \).

To reduce the neighborhoods of each vertex, we use a subgradient optimization procedure that gradually adjusts the vertex degree towards 2 for a minimum 1-tree by introducing a penalty to each vertex. In our experiment, the size of \( \mathcal{N}(x_i) \) is set to 15. The ReinforceNS algorithm utilizes these reduced neighborhoods for the subsequent search.

4.2 Initialization

To ensure efficient search, the quality of initial solutions is crucial for guiding the search towards promising regions. We apply TSP and KP approaches separately to generate an initial TTP solution \((\Pi, P)\) with high-quality.

We employ Chained Lin-Kernighan Heuristic [Applegate et al., 2003] (CLKH) to generate a high-quality route \( \Pi \). It is noteworthy that CLKH is one of the most widely used algorithms for solving TSP.

Based on the route \( \Pi \), we design a dedicated Iterated Greedy Packing (IGP) procedure to attain a good packing plan \( P \) for constructing an initial solution \((\Pi, P)\) for TTP. We adopt the scoring function as Eq. (4) [Faulkner et al., 2015] to evaluate item \( k \) picked up from city \( x_i \).

\[
s_{ik} = \frac{p_{ik}}{w_{ik} \times \xi_i} \tag{4}
\]

where \( \xi_i = d_{\pi}, x_i + \sum_{j=1}^{n-1} d_{x_j, x_j-1} \) represents the distance from city \( x_i \) to the ending city (depot) of the route. Notice that there is a trade-off between weight and profit value, we also introduce exponents to Eq. (5) and the function becomes as the following Eq. (5).

\[
s_{ik} = \frac{p_{ik}^\beta}{w_{ik}^\gamma \times \xi_i^\gamma} \tag{5}
\]

Different from [Faulkner et al., 2015], we use three parameters \( \kappa, \beta, \gamma \) to adapt the impact of profit value, weight and distance respectively.

The IGP procedure is composed of four steps. First, we randomly sample the values of \( \kappa, \beta, \gamma \) from a uniform distribution between 0 and 1. Second, iteratively choose an item with the highest score, calculated by Eq. (5), and pack it into the knapsack. When the total weight of items does not exceed the maximum weight \( W \) of the knapsack multiplied by a randomly selected factor \( \delta \) between 0.5 and 1, a feasible packing \( P_0 \) is obtained. Third, repeat the above two steps for \( \log(mn) \) times to determine a best-found packing \( P_0^* \). Finally, starting with the parameters \( \kappa, \beta, \gamma, \delta \) that attains \( P_0^* \),
we further explore the search for optimizing the packing by adjusting \( \kappa \) and \( \beta \), such as \((\kappa - \theta, \beta - \theta), (\kappa + \theta, \beta + \theta), (\kappa - \theta, \beta + \theta), \) and \((\kappa + \theta, \beta - \theta)\), where \( \theta \) is a parameter set to 5 and will progressively decreases by 0.75 times during the search. This process repeats 20 times, ultimately yielding a high-quality initial packing solution \( P \). Consequently, an initial solution \((\Pi, P)\) is effectively constructed.

### 4.3 Reinforcement Learning-Based Neighborhood Search

Neighborhood search is a powerful local optimization approach, we apply a reinforcement learning mechanism to speed up neighborhood examination, and propose a Reinforcement Learning-based Neighborhood Search (RLNS) procedure to discover better solutions.

Recall that \( L(T) \) is the length of the minimum 1-tree, and the \( \alpha \)-nearness \( \alpha(i, j) \) of an edge \((i, j)\) is the extra length of the \( L(T) \) to include this edge, we define the Q-value for city \( x_i \) and city \( x_j \) as shown in Eq. (6), where the parameter \( \tau \) is set to 0.5 and \( b \) is set to 1e-6.

\[
Q(i, j) = \frac{L(T)}{\tau \cdot \alpha(i, j) + (1 - \tau) \cdot d_{ij} + b}
\]  
(6)

The pseudocode of our RLNS is shown in Algorithm 2. After initializing the Q-value for each edge using Eq. (6), RLNS performs a series of iterations, with the maximum number of iterations \( \text{Iter}_{\text{max}} \) set to 1000. At each iteration, RLNS employs the \( \epsilon-greedy \) strategy for each city \( x_i \) (excluding the depot) to select a neighboring city \( x_j \) for 2-opt exchange. More precisely, it selects the best neighboring city \( x_j \) with the highest Q-value from the neighborhoods \( N(x_i) \) with a probability of \( 1 - \epsilon \), and randomly selects \( x_j \) with a probability of \( \epsilon = 0.2 \). Note that, the items of the sub-route from \( x_i \) to \( x_j \) should be removed from the knapsack of \( P \), since the sub-route between \( x_i \) and \( x_j \) is changed. For the cities located along the new sub-route from \( x_i \) to \( x_j \), RLNS adds an item with the highest score (Eq. (5)) among the items from these cities into the knapsack and repeats until the total weights exceeds the maximum weight of knapsack \( W \) multiplied by \( \delta \). The best local optimal solution is updated once a better objective value is found. Afterwards, the Q-value of \((x_i, x_j)\) is updated using Eq. (7). Here, the parameter \( \lambda \) and \( \phi \) are set to 0.1 and 0.9 respectively, and the state \( s_t \), action \( a_t \), and reward \( r(s_t, a_t) \) are defined as follows.

\[
Q(s_t, a_t) = (1 - \lambda) \cdot Q(s_t, a_t) + \\
\lambda \cdot \left[ r(s_t, a_t) + \phi \max_{a'} Q(s_{t+1}, a') \right]
\]  
(7)

- **State \( s_t \)**: The state at the current iteration \( t \) is a city \( x_i \) that is going to select a neighboring city for applying the 2-opt exchange operator.
- **Action \( a_t \)**: Select a neighboring city \( x_j \) of city \( x_i \) at the current iteration \( t \).
- **Transition**: The next state after performing the action is the next city that needs to select a neighboring city for the 2-opt exchange operator.

#### Algorithm 2 The pseudocode of the RLNS procedure

**Input:** The initial solution \((\Pi, P)\), the best objective value \( f' \)

**Output:** The local optimal solution \((\Pi', P')\)

1: Initialize Q-value of each edge by Eq. (6);
2: for \( \text{Iter} \leftarrow 0 \) To \( \text{Iter}_{\text{max}} \) do
3: Initialize the local objective value \( f_{\text{local}} = -\infty \);
4: for \( x_i \leftarrow 2 \) To \( x_n \) do
5: \( x_j = \epsilon-greedy(N(x_i)) \) \# Select \( x_j \) of the highest Q-value with a probability \( 1 - \epsilon \) and select \( x_j \) at random with a probability \( \epsilon \)
6: Remove items of \( P \) from the sub-route that begins at city \( x_i \) and ends at city \( x_j \) from the knapsack;
7: \( \Pi \leftarrow \text{Apply 2-opt exchange on} \Pi \text{at city} x_i \text{and} x_j \);
8: \( P \leftarrow \text{Iteratively add the items with the highest score as Eq. (5) of the sub-route from city} x_i \text{to city} x_j \text{into the knapsack without exceeding} W \times \delta; \)
9: if \( f(\Pi, P) \geq f_{\text{local}} \) then
10: \( f_{\text{local}} \leftarrow f(\Pi, P), \Pi_{\text{local}} \leftarrow \Pi, P_{\text{local}} \leftarrow \Phi \)
11: end if
12: Update Q-value of \((x_i, x_j)\) according to Eq. (7)
13: end for
14: if \( f_{\text{local}} > f' \) then
15: \( \Pi' \leftarrow \Pi_{\text{local}}, P' \leftarrow P_{\text{local}}, f' \leftarrow f_{\text{local}} \)
16: end if
17: \( \Pi \leftarrow \Pi_{\text{local}}, P \leftarrow P_{\text{local}} \)
18: end for
19: return local optimal solution \((\Pi', P')\)

- **Reward \( r(s_t, a_t) \)**: Suppose that the current solution \((\Pi_0, P_0)\), after performing the 2-opt exchange operator at city \( x_i \) and \( x_j \), the neighboring solution is \((\Pi_1, P_1)\). The difference between the objective values of the neighboring solution and the current solution is denoted as \( r(s_t, a_t) = f(\Pi_1, P_1) - f(\Pi_0, P_0) \).

During \( \text{Iter}_{\text{max}} \) consecutive iterations, the best-found solution \((\Pi', P')\) is updated if an improved solution has been found, and the best-found solution \((\Pi', P')\) is returned as the final output of RLNS.

### 4.4 The Post-Optimization Procedure

In addition to the RLNS procedure described above, the proposed ReinforceNS algorithm develops a post-optimization procedure to further improve the quality of the best-found solution \((\Pi', P')\) obtained by RLNS. This post-optimization is ensured jointly by an item flipping phase and a route refining phase.

The item flipping phase is devised to optimize the packing solution \( P' \) while keeping the route solution \( \Pi' \) unchanged. The flipping operator for an item involves removing it from the knapsack if it is already included or including it if not. During each iteration, the item flipping phase pseudo-performs the flipping operator for each item \( i \). If this operation leads to a new objective value, the item \( i \) is flipped using the item flipping operator. If no improvement is achieved, the item \( i \) remains unchanged. This process repeats until no new objective value is discovered during an iteration.
of the best-performing algorithms in the literature using three sets of 60 widely used benchmark instances.

ReinforceNS was implemented in C++ and compiled using GNU g++ 9.4.0 with ‘-O3’ optimization flag. The experiments were conducted on the AutoDL server with an Intel(R) Xeon(R) Platinum 8255C CPU@2.50 GHz and a memory limit of 40 GB. The maximum runtime limit \( T_{\text{max}} \) was set to 600 seconds, which is a common stopping condition for evaluating TTP algorithms. Given the stochastic nature of ReinforceNS, each instance was executed 10 times. Note that the source code of the reference algorithm CoCo [Namazi et al., 2023] is publicly available, we re-run CoCo on our platform to ensure a fair comparison.

**Benchmark instances.** The TTP benchmark library\(^1\), proposed by Polyakovskiy et al. [2014], comprises a total of 9720 instances with city counts ranging from 76 to 85900. In our experiments, we used a subset of 60 benchmark instances selected from this library, ranging from 76 to 33810 cities. These instances are widely used in previous studies and are categorized into three sets: CatA, CatB and CatC, each containing 10 specific instance files. We use suffixes to denote the specific selected instance files.

- **CatA:** The maximum weight of knapsack is relatively small. Each city has only one item. The weights and profits of the items are bounded and strongly correlated.
- **CatB:** The maximum weight of knapsack is moderate. Each city has five items. The weights and profits of the items are uncorrelated, and the weights of all items are similar.
- **CatC:** The maximum weight of knapsack is high. Each city has ten items. The weights and profits of the items are uncorrelated.

**Comparison with the best results from TTP competition.** We compare our ReinforceNS algorithm with the results of the most recent TTP competition organized by Markus Wagner et al. at the GECCO conference in 2023. Table 1 provides the comparative results for a total of 18 instances with the same time limit. These 18 instances are also classified into three sets: CatA, CatB and CatC. They are randomly selected from 10 instance files of each category, indicated by the suffix ‘-r’. The column “Competition” presents the best average results among 5 top-performing algorithms publicly reported on the website of GECCO 2023. The next column “ReinforceNS” reports our best average results over 10 runs. Note that we truncate the decimal of all results for brevity. The maximum weight of knapsack is high. Each city has five items. The weights and profits of the items are uncorrelated, and the weights of all items are similar.

- **CatA:** The maximum weight of knapsack is moderate. Each city has five items. The weights and profits of the items are uncorrelated, and the weights of all items are similar.
- **CatB:** The maximum weight of knapsack is high. Each city has ten items. The weights and profits of the items are uncorrelated.

**Comparison with three best-performing TTP algorithms.** Then we compare ReinforceNS with three top-performing

\(^1\)The TTP instances are available at https://cs.adelaide.edu.au/~optlog/CEC2014COMP_InstancesNew/
The minimum and maximum results among all the algorithms, respectively. The result of the reference algorithm, min and max are the algorithms from the literature: MATLS [Mei et al., 2014], S5 [Faulkner et al., 2015] and CoCo [Namazi et al., 2023]. Table 2 lists the best average results of these algorithms, consistent with previous studies. These 60 instances are selected from the first instance file of each category, typically used for comparing results of the TTP algorithms, indicated by the suffix “-1”. One easily observes that our ReinforceNS outperforms MATLS and S5 in terms of the best average result for all 60 instances. Additionally, ReinforceNS outperforms CoCo for 38 out of the 60 instances, especially for the two benchmark sets CatA and CatB with small and medium sizes.

The 95% confidence intervals based on the best average results in Table 2 are further introduced for a statistical evaluation of these algorithms. They are calculated using Eq. (9).

\[
95\% \text{ Confidence Interval} = \mu \pm 1.96 \times \frac{\sigma}{\sqrt{z}} \tag{9}
\]

Here, \( \mu \) represents the mean of the Relative Deviation Index \( RDI = \frac{\text{res} - \text{min}}{\text{max} - \text{min}} \times 100 \) [Kim and Kim, 1996], where res is the result of the reference algorithm, min and max are the minimum and maximum results among all the algorithms, respectively. The \( \sigma \) denotes the standard deviation of results, and \( z \) represents the number of instances. Figure 2 illustrates the 95% confidence intervals for these algorithms, highlighting that ReinforceNS outperforms the three comparison algorithms and demonstrates stable performance with the lowest variance.

**Ablation study.** To demonstrate the effectiveness of incorporating Reinforcement Learning (RL), we conduct an ablation study comparing ReinforceNS with RL against ReinforceNS without RL, denoted as ReinforceNS w/o RL. The
comparative results are summarized in Table 3. From the table, we can observe that ReinforceNS achieves superior performances, surpassing ReinforceNS w/o RL by 12, 12 and 15 instances in CatA, CatB and CatC, respectively. This suggests that the RL mechanism can indeed select promising neighboring solutions and facilitate an efficient search, leading to better results in most instances. Moreover, ReinforceNS with RL demonstrates competitive performance against ReinforceNS w/o RL, particularly on the large TTP instances. Additionally, Figure 3 shows that both ReinforceNS and ReinforceNS w/o RL demonstrate greater stability than the three top-performing TTP algorithms. Among them, ReinforceNS stands out as the most stable with the lowest variance.

### 6 Conclusion

In this paper, we propose the first learning-based algorithm utilizing the great power of reinforcement learning mechanism to combine with an effective neighborhood search heuristic for TTP. The proposed ReinforceNS algorithm incorporates a dedicated initialization procedure to identify promising regions, a neighborhood search procedure reinforced by a Q-learning strategy to find high-quality local optima, a post-optimization procedure for further improvement, and a multi-start mechanism to overcome deep local optima traps. We perform extensive experiments of the ReinforceNS on three sets of 60 benchmark instances. The comparisons with three state-of-the-art algorithms demonstrate the high competitiveness of our ReinforceNS in terms of solution quality and stability. In particular, ReinforceNS achieves improved results for 12 out of the 18 instances that were publishedly reported in the recent TTP competition at GECCO 2023. This work provides supporting evidence for using reinforcement learning to improve the efficiency of heuristics. As such it would be interesting to investigate other machine learning or deep learning techniques for solving TTP for future work.

### Acknowledgements

This work was supported in part by CCF-Huawei Populus Grove Fund and the National Natural Science Foundation of China (No. 62302174).

<table>
<thead>
<tr>
<th>Instance</th>
<th>CatA</th>
<th>CatB</th>
<th>CatC</th>
</tr>
</thead>
<tbody>
<tr>
<td>eil76-1</td>
<td>4425</td>
<td>13014</td>
<td>41079</td>
</tr>
<tr>
<td>kroA100-1</td>
<td>4565</td>
<td>13425</td>
<td>41284</td>
</tr>
<tr>
<td>ch130-1</td>
<td>9432</td>
<td>23590</td>
<td>75319</td>
</tr>
<tr>
<td>u159-1</td>
<td>9052</td>
<td>25830</td>
<td>772945</td>
</tr>
<tr>
<td>a280-1</td>
<td>18899</td>
<td>39479</td>
<td>143500</td>
</tr>
<tr>
<td>u574-1</td>
<td>27892</td>
<td>13057</td>
<td>13425</td>
</tr>
<tr>
<td>u724-1</td>
<td>51253</td>
<td>118434</td>
<td>391756</td>
</tr>
<tr>
<td>dsj1000-1</td>
<td>81167</td>
<td>221059</td>
<td>7681260</td>
</tr>
<tr>
<td>rl1304-1</td>
<td>101337</td>
<td>157509</td>
<td>526704</td>
</tr>
<tr>
<td>fl1577-1</td>
<td>165357</td>
<td>527866</td>
<td>1651740</td>
</tr>
<tr>
<td>d2103-1</td>
<td>268992</td>
<td>707586</td>
<td>6581260</td>
</tr>
<tr>
<td>pch3038-1</td>
<td>392257</td>
<td>1419201</td>
<td>9445360</td>
</tr>
<tr>
<td>fhl4461-1</td>
<td>713446</td>
<td>1938050</td>
<td>659610</td>
</tr>
<tr>
<td>pla7397-1</td>
<td>827741</td>
<td>2253510</td>
<td>19770750</td>
</tr>
<tr>
<td>rl11849-1</td>
<td>890845</td>
<td>2566320</td>
<td>1413741</td>
</tr>
<tr>
<td>usal13509-1</td>
<td>965523</td>
<td>2539980</td>
<td>5000960</td>
</tr>
<tr>
<td>brd14051-1</td>
<td>1096370</td>
<td>3041130</td>
<td>19770750</td>
</tr>
<tr>
<td>d15112-1</td>
<td>1793450</td>
<td>5935330</td>
<td>19699160</td>
</tr>
<tr>
<td>d18512-1</td>
<td>201337</td>
<td>621846</td>
<td>924266</td>
</tr>
</tbody>
</table>

Table 3: Comparisons between ReinforceNS with and without reinforcement learning

![Figure 3: The 95% confidence intervals for ReinforceNS w/o RL, ReinforceNS and three best-performing TTP algorithms (MATLS, S5, CoCo)](image-url)
References


[Polyakovskiy and Neumann, 2017] Sergey Polyakovskiy and Frank Neumann. The packing while traveling


