Alpha-T: Learning to Traverse over Graphs with An AlphaZero-inspired Self-Play Framework

Qi Wang (17110240039@fudan.edu.cn)
Fudan University School of Computer Science

Research Article

Keywords: Combinatorial Optimization, Deep Learning, Reinforcement Learning, Monte Carlo Tree Search

Posted Date: April 14th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-415344/v1

Alpha-T: Learning to Traverse over Graphs with An AlphaZero-inspired Self-Play Framework

Qi Wang*

School of Computer Science and Technology, Fudan University
* Corresponding author. E-mail address: 17110240039@fudan.edu.cn (Q. Wang).

Abstract. The combinatorial optimization problems on the graph are the core and classic problems in artificial intelligence and operations research. For example, the Vehicle Routing Problem (VRP) and Traveling Salesman Problem (TSP) are not only very interesting NP-hard problems but also have important significance for the actual transportation system. Traditional methods such as heuristics methods, precise algorithms, and solution solvers can already find approximate solutions on small-scale graphs. However, they are helpless for large-scale graphs and other problems with similar structures. Moreover, traditional methods often require artificially designed heuristic functions to assist decision-making. In recent years, more and more work has focused on the application of deep learning and reinforcement learning (RL) to learn heuristics, which allows us to learn the internal structure of the graph end-to-end and find the optimal path under the guidance of heuristic rules, but most of these still need manual assistance, and the RL method used has the problems of low sampling efficiency and small searchable space. In this paper, we propose a novel framework (called Alpha-T) based on AlphaZero, which does not require expert experience or label data but is trained through self-play. We divide the learning into two stages: in the first stage we employ graph attention network (GAT) and GRU to learn node representations and memory history trajectories, and in the second stage we employ Monte Carlo tree search (MCTS) and deep RL to search the solution space and train the model.

Keywords: Combinatorial Optimization, Deep Learning, Reinforcement Learning, Monte Carlo Tree Search.

1 Introduction

Combinatorial optimization is an important tool to solve some core problems in information theory, management science, computer science, artificial intelligence (AI), and other disciplines, and has extremely important applications in engineering technology, economy, military, and many other aspects [1]. Various algorithms and computational complexity theories to solve and explore combinatorial optimization problems have always been the focus of research in optimization theory, computer science, and other related fields. Known classic NP-hard problems such as the vehicle routing problem (VRP) [2], the traveling salesman problem (TSP) [3], and so on be-

long to the category of combinational optimization problems. Therefore, it is of great theoretical significance and practical application value to quickly solve large-scale combinatorial optimization problems.

Over the years, many traditional algorithms such as approximation algorithms and heuristic algorithms have been designed to solve combinatorial optimization problems, including the A* algorithm, simulated annealing algorithms, genetic algorithms, nearest neighbor algorithms, and heuristic solution solvers such as “OR-Tools”, “LKH3”, and “Concorde”. The above algorithms are often well targeted and accurate for specific problems, but for different instances of similar problems, we need to design new algorithms, again and again. That is, the previous solving experience is not helpful for the problems to be solved. Traditional algorithms do not make good use of the fact that in practical applications, most combinatorial optimization problems in the same scene have similar combinatorial structures and the differences are only in values and variables [4]. Therefore, researchers hope to design a general method that can excavate the essential information of the problem through learning, and improve the quality and efficiency of problem solutions by constantly updating the solution policy iteratively.

In recent years, with the rapid development of big data and artificial intelligence technology, machine learning [5], especially its deep learning [6] and RL [7] [8] is increasingly applied to solve combinatorial optimization problems [9] [10] [11]. In the face of huge search space and data points, it should be a reasonable scheme to combine the perceptual ability of deep learning with the reasoning ability of RL. Deep RL [7] has made a revolutionary breakthrough and extensive influence in the field of artificial intelligence, such as AlphaGo [12], AlphaGo Zero [13], and AlphaZero [14], etc. The fundamental motivation of applying deep learning and RL to combinatorial optimization lies in the discovery and reasoning of new policies. Compared with traditional algorithms, instance-based machine learning methods can discover the internal characteristics of instances by learning and apply the solving experience of existing instances to guide the solving of future instances, which also makes the solving of NP-hard problems that were not easy to solve in the past possible.

At present, the application of deep learning and RL in combinational optimization has obtained some preliminary results, but it is still in the experimental stage. Because the problems they solve, including VRP, TSP, MVC, and so on [4], are often limited to the scale of hundreds of nodes, and they need to adjust the network architecture, reward function, or decoding process to solve the same kind of problems with similar structure, which does not achieve complete universality and generalization. Most of the existing learning-based approaches are essentially learning heuristics [15], but they cannot have the ability to learn heuristics independently and require artificial design heuristics to assist learning. Besides, it remains a challenge to learn heuristics more cheaply, intelligently, and efficiently, and to integrate state-of-the-art models with their training methods, because the present RL has some problems such as sparse rewards, low sampling efficiency, and limited space exploration.

Combinatorial optimization is similar to Go in that they both seek feasible solutions or optimal solutions under constrained conditions in a huge combinatorial space. Even if in terms of search size, the combinatorial optimization problem on the graph

can have a much larger search space than go, so the combinatorial optimization on the graph is not lower in terms of complexity. AlphaZero won by thinking smarter rather than faster, by discovering the principles of board play on its own to develop a way of playing that reflects the truth of the game rather than programmer priorities and biases. It reveals the fact and advocates a direction that in the field of intelligent optimization we are not only competing for computing power, algorithms and scale, but also for intelligence and insights. Inspired by AlphaZero, we designed a lightweight framework for combinational optimization that applies the Q-Learning algorithm [16] with MCTS [17] to model the policy network and the value network. We first design a memory component which combines GAT [18] and GRU [19] to process the input graph, it can selectively memory state trajectories avoiding pre-training, and then we employ the history trajectories to the joint modeling policy and value, finally employ Q-learning algorithm with MCTS to optimize and update the policy network and value network.

To sum up, the main contributions of this paper are as follows:

- We design a memory component based on GAT and GRU, which can selectively remember historical trajectories for pathfinding and reasoning to omit the pre-training in previous work.
- We employ the history trajectories generated by the memory component to jointly model the policy network and the value network, which can be updated and promoted simultaneously.
- We employ the Q-learning with MCTS to uniformly optimize the policy network and value network, which is more efficient and more conducive to expanding the exploration space and avoiding the problem of sparse rewards than the REINFORCE algorithm or actor-critic algorithms [16].
- We design a scoring function based on MCTS to predict the nodes to be searched and reasoned about more efficiently in the testing stage.

2 Related Work

The application of deep neural networks in combinatorial optimization can be traced back to the Hopfield-network [20]. With the rapid development of artificial intelligence technology and hardware devices (such as GPU, TPU, etc.), more and more researchers are committed to applying machine learning [21] to traditional combinatorial optimization problems in recent years [22] [23]. Inspired by the great success of deep RL in games, some researchers have tried to transfer it to combinatorial optimization. To make this paper self-consistent, we first introduce the application of deep RL in games, then sort out the representative learning-based methods used in combinatorial optimization in recent years, and finally explain the inspiration of AlphaGo Zero for solving combinatorial optimization methods and the relationship between Go and combinatorial optimization problems.

2.1 Games with Deep RL

Deep RL [16], which integrates deep learning and RL [24], has become one of the most mainstream directions of artificial intelligence. Mnih et al. proposed DQN,

which combined with the deep neural network, Q-learning, and experience playback to achieve the human-level performance of Atari games [25]. DreamerV2 was the first agent to achieve human-level performance on 55 Atari benchmarks by learning behavior in a single trained world model [26]. Agent 57 was the first deep RL agent to exceed the standard human benchmark in all 57 Atari games [27]. DeepMind designed a series of algorithms for the more complex games of Go, chess, and so on, such as AlphaGo, AlphaGo Zero, and AlphaZero, and applied them to defeat professional human masters, officially ushering in the era of artificial intelligence [28]. MCTS [29] is one of the core techniques of these go/chess algorithms, which is applied to search for the huge state space while using UCB to make decision choices to balance exploration [17]. AlphaGo employs a two-stage training pipeline consisting of supervised learning and RL, expert experience-assisted training for a hot start in the initial phase, and then RL to iteratively update and optimize policy [12]. AlphaGo Zero and AlphaZero are updated versions of AlphaGo, which do not require any human experience but only the rules of go, then train themselves with the data generated by self-play and easily beat AlphaGo in tests [13]. Deep RL has been hugely successful in increasingly complex single-agent environments and two-player rotational games [30]. However, the real world is more complex and consists of multiple agents, each learning and acting independently and then cooperating and competing with each other. DeepMind applied tournament-style evaluations to verify that agents could achieve human-level performance in 3D multiplayer first-person video games such as "Quake III Arena in Capture the Flag Mode" [30]. DeepMind's Alpha Star was rated grandmaster in all three Starcraft tournaments, with over 99.8% of the officially ranked human players [31].

2.2 Combinatorial Optimization with ML/RL

We generally divide the learning-based methods into two categories: attention-based methods and GNN-based methods, although there are some overlaps and fuses between them.

Attention-based Methods. Inspired by the seq2seq model [32] in natural language processing (NLP), the pointer network [33] is proposed to solve the combinatorial optimization problem in a targeted way. It combines seq2seq and attention to deal with the feature that the length of the output sequence depends on the length of the input sequence in combinatorial optimization problems. The pointer network has changed the traditional attention, that is, the probability of each city in the input sequence will be obtained according to the attention when the output is predicted.

Bello et al. are the first to clarify that intensive learning is more suitable for combinatorial optimization problems than supervised learning, because it is expensive or even impossible to obtain lots of label data, and apply the actor-critic algorithm [16] to train the Pointer Network to solve the TSP problem [34].

Based on [33] and [34], Nazari et al. proposed an end-to-end RL method to solve VRP, which simplifies the pointer network and can effectively handle both static and dynamic elements. It extends the Pointer Network to both VRP and TSP and exceeds traditional heuristics and OR-tools [35].

Kool et al. built a framework based on the transformer using only the attention mechanism [36] to solve the combinational optimization problem [37]. Instead of using the sequence as input, it uses the graph as input, which eliminates the dependence on the order of the nodes in the input [18]. That is, no matter how we arrange the nodes, the output will not change if the given graph does not change, which is an advantage over the sequential approach.

GNN-based Methods. As a generalization of deep learning in graph data, graph neural network (GNN) [38] [39] has been widely used in combinatorial optimization in recent years due to its strong ability to represent the intrinsic structure of graphs and its ability to aggregate information of nodes and edges.

Dai et al. proposed a scheme of finding evaluation function with graph embedding [40]. They first applied Structure2Vec to embed the graph into the low-dimensional space, and then employed the greedy Q-learning algorithm, which was based on the goal of maximizing an "evaluation function" and applied the method of adding nodes step by step to get the solution of the problem [4].

Li et al. applied supervised learning training GCN [41] to guide the parallel tree search process, which quickly generated lots of candidate solutions and selected one of them after subsequent optimization [42]. Mittal et al. designed a two-stage training method based on [4] and [42], in which supervised learning followed by RL and greedy probability distribution were applied to solve combinatorial optimization problems on large-scale graphs [43]. The methods for combinatorial optimization which are designed into an encoder-decoder paradigm based on GNN framework also include [44] [45] [46] [47].

Chen et al. proposed a neural writer that learns a policy to select heuristic algorithms and rewrites local components of the current solution to iteratively improve it until it converges [48]. Lu et al. proposed "L2I", which starts from a random initial solution and learns to iteratively improve the solution with an improved operator selected by the controller based on RL [49].

2.3 AlphaGo Zero's Enlightenment on Combinatorial Optimization

AlphaGo Zero [13] only applies one deep neural network to replace the two independent networks in AlphaGo [12], namely the policy network and the value network and outputs the playing policy and the winning rate value of the current chess board situation, which not only saves the computing space and reduces the operation consumption, but also the mixed two networks can better adapt to a variety of different situations.

Games vs. Combinatorial Optimization. (1) Combinatorial optimization also has a huge solution space like Go. For example, the solution set of TSP with n cities can reach an order of magnitude of $O((n-1)!)$. Therefore, we can also establish an appropriate mathematical model for combinatorial optimization problems and adopt the MCTS to reduce the solution space. (2) Similar to the winning and losing rules in Go, combinative optimization problem also has clear objective function and constraint conditions (rules) to evaluate current policy and learn from reward signals. (3) The heuristic function designed based on expert experience or the model trained with a small amount of label data may be locally optimal in the convergence domain, while

we can achieve self-optimization by generating data and high-quality sample labels by self-play like AlphaGo Zero (or AlphaZero).

At present, some scholars have been inspired by the success of AlphaGo Zero in Go and have transferred it to combinatorial optimization to achieve higher accuracy and efficiency [50] [51] [52] [53]. However, a common problem of these methods is the excessive demand for experimental environment. Besides, the policy or value network in the previous MCTS-based methods were trained independently without MCTS participation and only applied to assist MCTS after training, while our method improved the policy by learning the trajectory generated by MCTS and using the information of the entire MCTS search tree.

3 Methodology

3.1 Problem Setup

VRP is a generic term for a class of problems in which a fleet based on one or more warehouses must determine a set of routes for a geographically dispersed city or customer [2]. The goal of VRP is to provide a set of customers with known requirements with the lowest-cost vehicle routes starting and ending in a warehouse (Figure 1). TSP is a simple special case of VRP that describes starting from one city and returning to the starting point after traversing all the other cities with the lowest total cost [3]. VRP is one of the most challenging combinational optimization (NP-hard) problems, and much of the interest over the years stems from its practical significance and its difficulty.

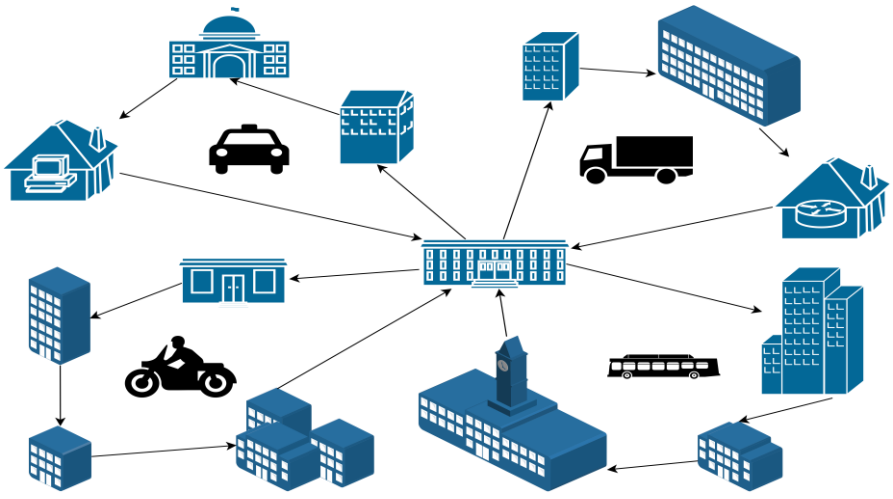

Fig. 1. The demonstration of VRP in the practical application scenario, in which each node can be different facilities or cities containing their demand; Different paths correspond to different miles; A vehicle can be a truck, a motorbike, a taxi or a shuttle bus with their respective load capacity. The goal of VRP is to optimize a set of paths to minimize the total cost.

We formally define VRP and TSP on the graph, and because we employ the MCTS algorithm in the framework, the optimal path problem on the graph can be translated into searching for the path with the minimum cost in the tree [51]. $G = (V, E, W)$ is used to represent an undirected and unlabeled graph, where V is a set of vertices and E represents each node and the optimal path is composed of a sequence of nodes $\{v_0, \dots, v_i, \dots, v_0\}$ where v_0 represents the starting node, E is a set of edges, and W is a set of weights on the edges $e_{ij} = (v_i, v_j)$.

VRP is Converted to TSP. In theory, we can apply distributed multi-agents to simulate multi-vehicles to find the paths on the graphs. That is, each agent represents a vehicle. However, due to our limited implementation capacity, computing resources, etc., we are still committed to designing a generic and lightweight framework for finding and reasoning paths. If we apply the subgraph sampling algorithm [54] to divide the VRP on the graph into subgraphs of the same size, and each subgraph contains a central node, then the VRP can be converted to TSP. In this way, we can solve both VRP and TSP without defining two Markov decision processes for them respectively, and subgraph sampling is equivalent to greedy probability distribution [43] and Graph reduction [42] to effectively preprocess large-scale graphs.

Heuristic Functions. The learning-based method to solve combinatorial optimization problems is essential to learn the corresponding heuristics, so we can construct a heuristic function for the combinatorial optimization problems on the graph to guide the learning. At the same time, inspired by the Q&A system [55], we can also turn the combinatorial optimization problem into finding an answer to a certain question. For example, for TSP, we can construct a function $f(G, v_0, q)$, whose functional form is generally unknown, where q is a relational query or target such as "shortest distance", "longest distance", and "lowest cost", etc. We need to use something like $(v_0, \dots, v_0)$ such samples (the optimal solution sets) to constitute a training data set to learn $f(G, v_0, q)$. Therefore, in theory, we can solve other combinatorial optimization problems [42], such as Minimum Vertex Cover (MVC), Maximum Cut (MC), and Maximal Independent Set (MIS), by setting different relational queries and targets to make the heuristic function $f(\cdot)$ more universal.

3.2 Markov Decision Process (MDP)

In this paper, $f_\theta(G, v_0, q)$ is modeled by a graph walking agent, which can intelligently traverse all the remaining nodes in the graph and return to the starting point, and requires the shortest sum of traversed paths. The agent must learn a search policy from the training set so that when the training is complete, the agent understands how to traverse all nodes in the graph with minimal cost. The agent is not supervised from beginning to end but receives only delayed evaluation feedback: the agent is rewarded positively when it correctly predicts the optimal path in the training set. For this purpose, we formulate the optimal path problem as a MDP so that we can train the agent through RL.

The MDP is defined by the tuple (S, A, P, R), where S is the set of states, A is the set of actions, P is the state transition probability and R is the reward function. Our environment is a limited range, deterministic, and partially observed GAT located on a VRP graph.

Actions. Action space A refers to all actions performed by the agent in the graph. Every time the agent selects a node v_i, it means that it executes an action a_t $(a_t \in A)$ in the time step t.

States. S represents the state space, and s_t represents the node set traversed by the agent up to time step t. At s_t, the agent can perform two actions: (1) select an edge e_{ij} and move to the next node. (2) Terminate the walk after returning the start node v_0 (the state becomes s_{end}). The agent needs to make a correct decision based on the traversed node sequence trajectory and the target, so we recursively define s_t:

$$s_t = s_{t-1} \cup \{a_{t-1}, v_{i,t}, N_{v_{i,t}}, E_{v_{i,t}}\} \quad (1)$$

Where $v_{i,t}$ is the node visited by the agent in time step t, $N_{v_{i,t}}$ represents the set of adjacent neighbor nodes with $v_{i,t}$, and $E_{v_{i,t}}$ represents the set of edges connected with $v_{i,t}$.

Transition. P refers to the probability that the state changes after an action is performed. $P(s_{t-1}, a_{t-1}, s_t) = 1$ if and only if s_t represents a partial tour produced by adding $v_{i,t}$ to the partial tour in s_{t-1}.

Rewards. We define the reward function $R(s)$ as the length of the tour, if $s_t = s_{end}$ and $a_t = v_{0,t}$ (represents the complete tour), otherwise $R(s)$ =0.

3.3 Memory Components with GRU Encoder and Graph Attention

In order to solve the finite time domain deterministic part of the observable GAT, we introduce the GRU [19] to memorize previously traveled paths (states or actions) for learning random history-dependent policies instead of using supervised learning in AlphaGo [12] and GCOMB [43] for pre-training. Pre-training with supervised learning requires many known optimal paths for model training, and such “hot starts” may cause the model to overfit on given paths.

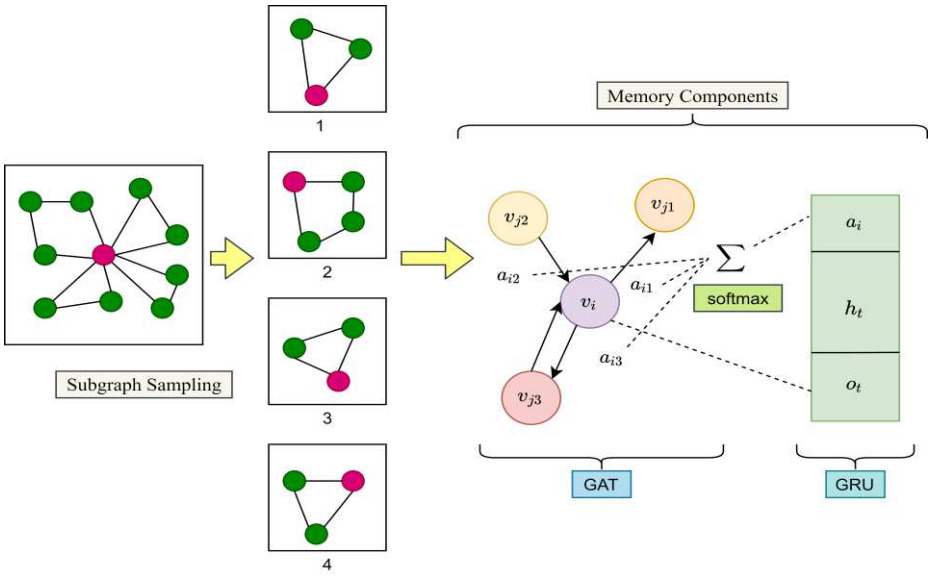

Fig. 2. Flowchart of memory components with GAT and GRU. We take the graph as input and apply subgraph sampling to divide it into k subgraphs with central nodes, then iteratively process each subgraph using the memory component. The subgraph sampling process in the figure is just a simple example, while the actual graph is much more complex. The memory component combining GAT and GRU can effectively infer the importance of neighbor nodes and remember and update their status and trajectories, which enables the model to avoid pre-training.

The agent-based GRU encodes the state s_t as a continuous vector h_t, $h_t = ENC(s_t)$, and the history embedding can be dynamically updated according to the GRU:

$$h_t = GRU(h_{t-1}, [a_{t-1}; o_t]) \tag{2}$$

Where o_t represents a vector representation of an observation, and $[;]$ denotes vector connection.

Based on historical embedding h_t, the policy network decides to take an action from all available actions A based on the query q. Each possible action represents a node or an outgoing edge with labels and information. In order to take the graph as the input and enable the agent to have a larger field of vision during traversal to better judge the weights of paths and nodes, we introduce the GAT [18] to process the information on the graph.

We employ the complete graph as input rather than a sequence of nodes because we can apply GNN to learn the internal structure of the graph so that we can aggregate and update the information on the graph even when the nodes and edges in the graph are changing dynamically (figure 1). As an encoder of the graph, we first obtain the original feature of the node on the graph $e_i = Embed(v_i), e_i \in R^F$, where F is the dimension of the feature vector, and $Embed()$ can be a fully connected neural network such as FCN, MLP, etc. Then we apply each layer of the GAT to update the

feature vector of the $i-th$ node, and the attention weights from node i to node j are calculated as follows:

$$a_{ij} = a(We_i, We_j) \tag{3}$$

Where a is a mapping of $R^{F'} \times R^{F'} \to R$, and $W \in R^{F' \times F}$ is a weight matrix. To ensure that the structural information on the graph is not lost, we apply masked attention which simply assigns attention to the adjacent node set of the node v_i. Normalized self-attention is shown as follows:

$$\alpha_{ij} = softmax_j(a_{ij}) = \frac{\exp(a_{ij})}{\sum_{k \in N_i} \exp(a_{ik})} \tag{4}$$

We employ a single layer neural network a, and the specific calculation process is as follows:

$$\alpha_{ij} = \frac{\exp(LeakyReLU(\vec{a}^T[We_i || We_j]))}{\sum_{k \in N_{v_i}} \exp(LeakyReLU(\vec{a}^T[We_i || We_k]))} \tag{5}$$

Where $\vec{a}^T \in R^{2F'}$ is the parameter of the feedforward neural network a and $LeakyReLU$ is the activation function. We can get the updated vector:

$$e_i' = \sigma(\sum_{j \in N_{v_i}} \alpha_{ij} We_j) \tag{6}$$

To improve the representational ability of the model, we apply multiple W^k to calculate self-attention at the same time, and then combine the results obtained by each W^k:

$$e_i' = ||_{k=1}^{K} \sigma(\sum_{j \in N_{v_i}} \alpha_{ij}^k W^k e_j) \tag{7}$$

Where $||$ represents the connection, and α_{ij}^k and W^k represent the calculation result of the $k-th$ head. We can also take the summation to get e_i':

$$e_i' = \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in N_{v_i}} \alpha_{ij}^k W^k e_j) \tag{8}$$

We can also replace GAT with the state-of-the-art DAGN [56], which is a principled approach that incorporates multi-hop adjacent contexts into attention calculations, supporting remote interaction at each level. The state vector of the node v_i at the time t can be calculated using the following formula:

$$s_{i,t} = [[a_{t-1}; o_t]; h_t; e_i'] \tag{9}$$

3.4 Joint Modeling Policy and Value with Self-Play Neural Network

We apply both the policy $\pi_\theta(a_t|s_t)$ and the Q_θ, where θ is the model parameter. $\pi_\theta(a_t|s_t)$ represents the probability that the agent performs the action a_t in the current state s_t, which is regarded as a prior to bias of the MCTS. Q_θ defines the long-term reward obtained by following the optimal policy and taking the action a_t at the state s_t. We aim to learn a policy that can maximize long-term rewards so that the agent can predict nearby target nodes with high probability and eventually return to the original node to form a tour (Figure 3 shows the complete process of our modeling and training).

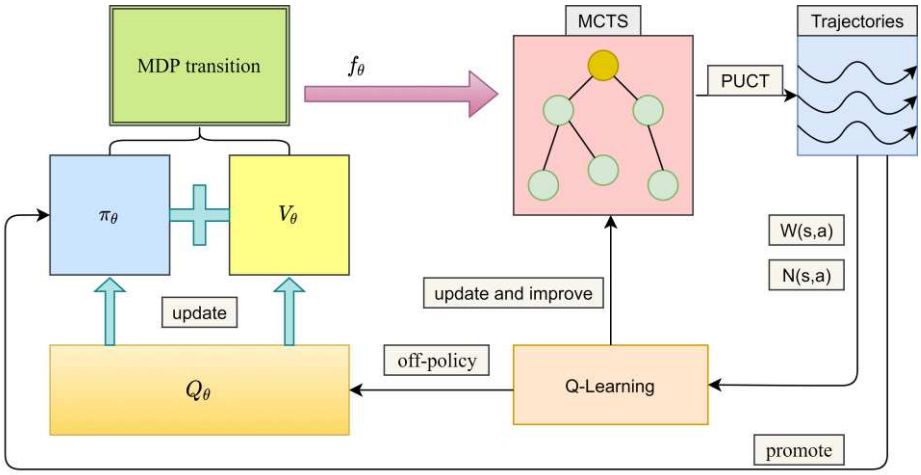

Fig. 3. Q-learning with MCTS is applied to simultaneously model and train the policy network and the value network.

As described in Section 3.3, h_t is composed of the following parts: vector $h_{S,t}$ encodes $(s_{t-1}, a_{t-1}, v_{i,t})$, i.e., the state, action, and current node at the last time; $h_{v_{i,t}',t}$ $(v_{i,t}' \in N_{v_{i,t}})$ integrates neighbor nodes $N_{v_{i,t}}$ and edges $E_{v_{i,t}}$; $h_{A,t}$ judges whether the agent arrives at the destination based on $N_{v_{i,t}}$ and $E_{v_{i,t}}$. We can model π_θ and Q_θ by the following formula:

$$x_0 = f_{\theta_\pi}([h_{S,t}, h_{A,t}]) \tag{11}$$

$$x_{v_{i,t}',t} =< h_{S,t}, h_{v_{i,t}',t} > \tag{12}$$

$$V(s_t) = Q_\theta(s_t, \cdot) = Sigmoid(x_0, x_{v_1}', \dots, x_{v_m}') \tag{13}$$

$$P(s_t, a_t) = \pi_\theta(\cdot \mid s_t) = Softmax_\tau(x_0, x_{v_1}', \dots, x_{v_m}') \tag{14}$$

Where x_0 is obtained by stitching $h_{S,t}$ and $h_{A,t}$ together through a full-connected neural network $f_{\theta_\pi}()$. $x_{v_{i,t}',t}$ is given by the inner product of $h_{S,t}$ and $h_{v_{i,t}',t}$. x_{v_m}' refers to the neighbor's score, Q is obtained by the $Sigmoid$ function, and π is obtained by the $softmax$ function with the temperature parameter τ [57].

3.5 Training Algorithm with MCTS

Inspired by AlphaGo [12], we design a training pipeline to train the parameter θ from beginning to end, and we further define $f_\theta(s_t) = (P(s_t, a_t), V(s_t))$ according to [13] [28], where $P(s_t, a_t)$ is the action probability, and $V(s_t)$ is the evaluation score of the model predicted by the network f with parameter θ for the given data set D, task T_a and state s_t. The neural network predicts the probability of taking an action, which in turn addresses a given task on a data set. The network input is a self-playing training instance (s_t, π_θ, z_t), where z_t is the evaluation of the pipeline at the end of the traversal. The parameter θ of network f is adjusted by the stochastic gradient descent (SGD) of loss function L, which calculates the sum of the mean squared error and the cross-entropy loss:

$$L(\theta) = (z - V)^2 - \pi^T log P + c\|\theta\|^2 \tag{15}$$

The L2 regularization parameter c is used to prevent overfitting. The network output is the probability of action $P(s_t, a_t)$ and the evaluation $V(s_t)$ of pipeline performance. A little bit different from the loss in [28] is the normalized cumulative reward z that we use. Instead of comparing two players' odds of winning, the model evaluator generates random graph instances each time and compares their average performance, specifically:

$$r_e(s_t) = \begin{cases} 0, s_t = s_{end} \\ \mu'_s + \epsilon_s \cdot (max_{a \in A} V) \end{cases} \quad (16)$$

When estimating the state value of state s_t, we perform an action to a_t maximize $V(s_t)$ and apply ϵ_s and μ'_s to recover the unnormalized value, where ϵ_s and μ'_s are the standard and the mean deviation of the cumulative rewards of the random plays from state s_t.

Our algorithm runs multiple simulations in the MCTS [17] through neural network predictions $(P(s_t, a_t), V(s_t))$ using them to search for better evaluation. The search results improve the predicted results given by the network by using update rules to improve network policy:

$$U(s_t, a_t) = Q(s_t, a_t) + \gamma P(s_t, a_t) \frac{\sqrt{N(s_t)}}{1 + N(s_t, a_t)} \quad (17)$$

Where, $Q(s_t, a_t)$ is the expected reward of an action a_t starting from state s_t, $P(s_t, a_t)$ is the neural network's estimation of the probability of an action a_t starting from state s_t, $N(s_t, a_t)$ is the number of an action a_t starting from state s_t, $N(s_t)$ is the number of times to access state s_t, and γ is the constant to determine the amount of exploration. At each step of the simulation, we need to find the action a_t and state s_t that maximize $U(s_t, a_t)$, and add the new state to the tree if it does not exist in the neural network estimation $(P(s_t, a_t), V(s_t))$, otherwise the search is called recursively.

We developed a new reverse link algorithm that employs MCTS to leverage MDP transfer functions. That is, in each MCTS simulation, the trajectory is expanded by selecting actions based on variations of the PUCT algorithm [13] from the root state s_0:

$$a_t = argmax_a \{\beta \cdot \frac{\pi_\theta(a|S_t)^\mu \sqrt{\sum_{a'} N(s_t, a')}}{1 + N(s_t, a)} + \frac{W(s_t, a)}{N(s_t, a)}\} \quad (18)$$

Where β and μ are constants that resemble γ.

The core idea of Alpha-T is to run multiple MCTS simulations to generate a set of trajectories with more positive rewards, which can be understood as being generated by a promoted policy π_θ, while learning these trajectories can in turn promote the policy. Since these trajectories are discrete rather than continuous, we employ Q-learning [58] [59] instead of REINFORCE [16] to update Q-network from these trajectories in an off-policy way:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta Q_\theta(s_t, a_t) \times (R(s_t, a_t) + \varepsilon max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t)) \quad (19)$$

Since the Q-network (value Network) and the policy network share the same set of parameters, as soon as the parameters in the Q-network are updated, the policy network will also be improved automatically at the same time, and the new policy will be applied to control the MCTS in the next iteration.

According to [13], the data generator repeatedly generates random graph instances as self-playing records based on the current best model of MCTS, which is referred to (s_t, π_θ, z_t) as mentioned above, indicating that taking actions from the states depends on the policy of MCTS-improved. The learner randomly samples and updates the parameters of the best model from the generator's data (the above Q-learning update process with MCTS). The description of the training algorithm with MCTS is shown in Algorithm 1.

Algorithm 1 Q-learning with MCTS

Input: Graph G, Network f_θ, initialize parameters θ_0 of the network f_{θ_0}, a mini-batch size b, root (initial) node (state) s_0, query q, MCTS search number T, Maximum time steps T_{max};

for episode $= 1, \ldots, T$ **do**
  for $t = 0, \ldots, T_{max}$ **do**
    while s_t is expanded before and $s_t \neq s_{end}$ **do**
      Perform a MCTS guided by f_θ
      Obtain $W(s_t, a)$ and $N(s_t, a)$

$$a_t = argmax_a\{\beta \cdot \frac{\pi_\theta(a|s_t)^\mu \sqrt{\sum_{a'} N(s_t, a')}}{1 + N(s_t, a)} + \frac{W(s_t, a)}{N(s_t, a)}\}$$

      Take the action a_t , observe next state s_{t+1}, and update f_θ
    end while
      $\{expand\}$
    if $s_t \neq s_{end}$ **then**
      Compute estimated reward value $V_\theta(s_t) = Q(s_t, a_t)$
      Add the generated node v_i to the partial tour
      $\{backup\}$
      $r = r_e(s_t)$
    while $s_t \neq s_0$ **do**
      $a = previous\ action$
      $r \leftarrow r + R$
      $r' = (r - \mu'_s)/\epsilon_s$
      $W(s_t, a) \leftarrow W(s_t, a) + r'$
      $N(s_t, a) \leftarrow N(s_t, a) + 1$
      $Q(s_t, a) \leftarrow \frac{W(s_t, a)}{N(s_t, a)}$
      Break
    end if
  end for
end for
for each node is added to the tour **do**
  if $s_t = s_{end}$ and $a_t = v_{0,t}$ reward $R = 1$, otherwise $R = 0$
  Q-learning is applied to update model parameters
$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta Q_\theta(s_t, a_t) \times (R(s_t, a_t) + \varepsilon max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t))$

end for

3.6 Predict Nodes with the MCTS Scoring Function

In the test phase, we generally apply the trained policy to predict the sequence of nodes (paths) on the unseen (unknown) graph. Previous methods predict the probability distribution of nodes on the unknown graph, such as the greedy probability distribution scoring function used in GCOMB [43], while our method combines the policy and value learned by MCTS to generate an MCTS tree during the training phase. In the testing phase, previous methods often predict multiple paths based on different rewards, but in practice, we often need an accurate shortest path, which means that the result we need should be unique. When alpha-T predicts multiple paths, leaf states in the MCTS tree correspond to nodes on different paths, and we need to combine the predicted results of MCTS leaf states into a score to sort the nodes, specifically:

$$Score(v_i) = \sum_{s \to v_i} \frac{N(s,a)}{N} \times Q_\theta\,(s, v_{0,t}) \tag{20}$$

Where N is the sum of all leaf states s corresponding to the same node v_i. $v_{0,t}$ is an action in the terminated state (see section 3.2). $Score(v_i)$ is the weighted average value of terminal state values associated with the same candidate node v_i. Among the candidate nodes (paths) we choose the one with the highest score:

$$v_p = argmax_v Score(v_i) \tag{21}$$

4 Experiments

4.1 Data Sets and Settings

Alpha-T does not require expert experience to learn heuristics through self-play, while data can be generated by data generators. In this paper, we focus on path optimization problems, and we can generate VRP or TSP instances to train the model as in previous works [37]. Our method is based on MCTS and RL, which theoretically makes it more generalization than supervised learning (which relies on label data) and has more search space than other search methods. Therefore, we can also generate ER and Barabasus-Albert (BA) graphs to extend the model to other similar combinatorial optimization problems on a larger scale [43] [4].

In terms of baseline selection, we chose some classic heuristics, the most recent ones based on attention or GNN, and AlphaGo Zero-based methods. The experimental results of baselines in the comparison experiment were taken from their respective papers. Due to the difficulty of reproducing the previous work (such as computing resources and time), we chose the experimental tasks and parameter settings similar to previous works as far as possible for convenience comparison.

For the memory component, we set GRU hidden dimension to 200, the attention dimension to 100, MLP contains 5 layers and each layer hidden dimension is set to 100. For MCTS we follow AlphaZero, AlphaGo Zero, and ELF OpenGo [60], we set the temperature parameters $\tau = 1$ in the training phase, while in the test phase $\tau = 0$. During the training, each learner sampled 20 tracks from the self-play record and

conducted random gradient descent through Adam, with a learning rate of 0.001, a weight attenuation of 0.0001, and a batch size of 16. The learner saves the new model after 15 iterations. We set the number of data generators, learners, and model evaluators to 20,6 and 2, respectively. Meanwhile, we incorporate PBT [61] to optimize the parameters in the network, which is a population-based method for accelerating and improving AlphaZero. The experiments are performed on eight NVIDIA GeForce GTX1080Ti GPUs, and part of the code is available at:
https://github.com/wangqi798252101/Alpha-T

4.2 Performance for Small-Scale TSP

We first verify the efficiency and effectiveness of Alpha-T on small TSP instances. The selected baselines are representative GNN-based methods, including GAT, GCN, Structure2Vector, and Graph-Pointer network, as well as traditional heuristic search algorithms such as Nearest and MST [37] [47]. The optimal solutions of TSP20/50/100 instances are relatively easy to be obtained by the solution solvers (such as Concorde, LKH3, and OR Tools [37]), and the approximate ratio of each method to the optimal solution is applied to measure the quality of the solution obtained by the model, among which the method with the lower approximate ratio is the better.

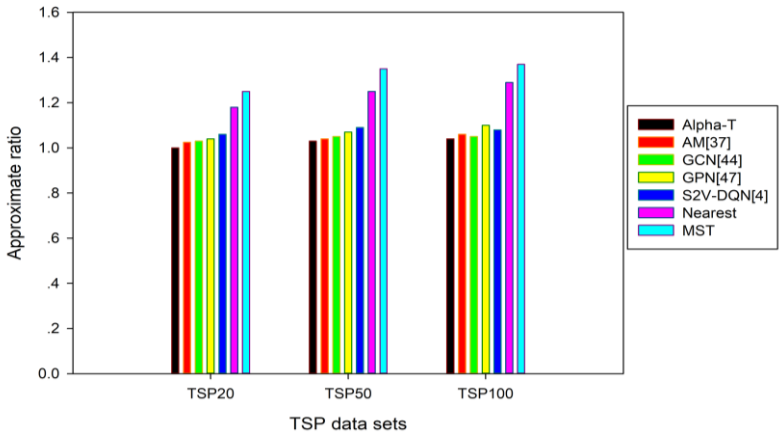

Fig. 4. The approximate ratio of the solutions generated by different methods on small TSP instances to the optimal solutions.

Figure 4 shows the experimental results of different methods. From the trend, we can see that the solution obtained by Alpha-T is the closest to the optimal solution, which indicates its good effect on small-scale path problems. We think that compared with the previous method based on GNN alone, Alpha-T applies GAT with GRU, which effectively integrates and remembers the information of neighboring nodes to form a more complete state space and action space to guide agents to better find nodes and reasoning paths. During the experiment, we compared the memory components with GAT alone, and we found that using GRU to aid attention on a small scale is more accurate than just using attention. We believe that attention can infer the im-

portance of the surrounding neighbors so that the GRU can record the historical trajectories more efficiently and avoid a lot of redundant states and actions. Moreover, if the agent can remember the paths taken, it is helpful to infer the paths to take.

4.3 Performance for Larger-Scale TSP

Real-world transportation networks often contain hundreds of nodes, so it is necessary to verify alpha-T's performance on a larger TSP instance. We train on small scale graphs and then we reason on larger scale graphs, which is another way to demonstrate the generalization ability of the model. Since the GNN-based method may be too heavy to be trained on the big picture due to too many parameters, this phenomenon is common in the previous methods [4]. At the same time, due to the Alpha-T framework using GRU, which is a variant of RNN, if all training in large scale figure could lead to a gradient or explosion due to excessive parameters is too heavy to be trained, so for the super-large size graphs, we can apply the subgraph sampling algorithm [54] to divide into subgraphs and iteratively deal with these subgraphs [62]. If there is too much training data, the training time will be too long or we do not know how long the training time is appropriate, because if the training time is too long, it will lead to overfitting, while the training time is short, the potential of the model cannot be fully released. To sum up, we trained for one hour on TSP20, TSP50, and TSP100 respectively, and then tested on TSP500, TSP750, and TSP1000. Table 2 shows the comparison results with baselines.

Table 2. The model performance of Alpha-T trained on small TSP instances and then reasoned on larger instances compared to the results of the baseline approaches. The indicators measured are the optimal solution obtained by the model and the time required to obtain it when reasoning.

Method	TSP250	TSP750	TSP1000
	Obj. Time	Obj. Time	Obj. Time
Concorde	11.89, 1894s	20.10, 32993s	23.11, 47804s
LKH3	11.893, 9792s	20.129,36840s	23.130, 50680s
OR Tools	12.289, 5000s	22.395, 5000s	26.477, 5000s
Nearest Insertion	14.928, 25s	25.219, 115s	28.973, 136s
2-opt	13.253, 303s	22.668, 3296s	26.111, 6153s
Farthest Insertion	13.026, 33s	22.342, 454s	25.741, 945s
GPN	12.942, 214s	22.541, 2278s	26.129, 4410s
S2V-DQN	13.079, 476s	22.550, 3182s	26.046, 5600s
AM	14.032, 2s	28.281, 42s	34.055, 136s
Alpha-T(TSP20)	**12.857, 4s**	**22.524, 35s**	**25.698, 120s**
Alpha-T(TSP50)	**12.798, 4s**	**21.985, 33s**	**24.856, 115s**
Alpha-T(TSP100)	**12.783, 3s**	**21.864, 32s**	**24.451, 109s**

From the experimental results, we can see that Alpha-T achieves better results compared with the previous RL methods, but it is still not as of accurate as the tradi-

tional heuristic methods. We can also observe that Alpha-T performs better when the training data set is larger, which indicates that the increase of training data within a certain range is conducive to improving its ability and unleashing its potential. To analyze the reason, we applied the memory component in the framework, which is helpful for the agent to find the correct node more accurately. Moreover, the Q-learning algorithm based on MCTS can further expand the search space and search efficiency, because it enhances the sampling efficiency and solves the problem of reward sparseness.

4.4 Performance on VRP

In theory, we can apply Alpha-T to solve other types of combinational optimization problems on graphs by simply changing the GAT to fit other problems such as MVC, graph coloring, maximum cut problems, and so on [50] [52]. As mentioned above, this paper focuses on VRP on the graph, which can be transformed into TSP through subgraph sampling algorithm. In the VRP experiment, k is set as 5, that is, the VRP instance in the experiment is converted into 5 TSP instances. The model trained in TSP20 is used for VRP100, TSP50 for VRP200, and TSP100 for TSP400. We try to establish a method system to solve these kind of path problems based on AlphaZero, so whether good effects can be maintained on other problems will be reserved for future work. The previous works based on AlphaGo Zero are mostly used in graph coloring problems, 3D packing problems, MVC, etc. [50] [53] [52] [51], which did not overlap with our methods in the experiment, and such methods were often too complex to be easily repeated, so we still chose the targeted method to solve VRP as the baselines.

From the experimental results, we can see that Alpha-T is still very competitive compared with the baseline, which mainly proves the effectiveness of the subgraph sampling algorithm for the conversion of VRP to TSP because we have previously proved the efficiency of Alpha-T on TSP. We could have handled VRP directly by changing the GAT without the need for transformation, but we try to unlock the possibility of fast parallel processing of large-scale graphs. In the combinational optimization problems, a lot of problems have mathematical equivalence, for example, Independent Set (MIS), Minimum Vertex Cover (MVC), and Maximal Clique (MC) in the bipartite graph [42]. We can apply machine learning to convert them to each other to make it possible to solve multiple problems using only one model that does not need to be modified. Besides, we apply Q-learning with MCTS to achieve more efficient learning heuristics, search through trees and graphs, and update the network so that the agent can find the target node more intelligently.

Table 3. The performance of Alpha-T in the VRP instances. We employ the solution solver OR-Tools as a benchmark to calculate the performance of each model relative to it, where "Obj" represents the length of the optimal solution obtained, and "Gap" represents the Gap between each model and "OR-Tools". To facilitate the comparison, we choose the most representative methods recently, among which the performance of the methods with different search algorithms is different, and we choose the most effective version.

Method	VRP100	VRP200	VRP400

	Obj	Gap	Obj	Gap	Obj	Gap
OR-Tools	11.348	0.00%	17.995	0.00%	26.095	0.00%
PRL [35]	11.907	4.93%	19.521	8.48%	28.846	10.54%
AM [37]	11.746	3.51%	18.697	3.90%	27.143	4.02%
GCN-NPEC [46]	11.186	-1.43%	17.516	-2.66%	25.876	-0.84%
Alpha-T	**11.271**	**0.68%**	**18.457**	**2.57%**	**26.642**	**2.10%**

4.5 Effect of Training

So far, there are still some problems in the application of RL, such as sparse rewards, low sampling efficiency, limited search space, etc., because, under normal circumstances, agents may find a lot of useless nodes or paths in constant trial and error, or enter an endless cycle in a certain state. Even the role of RL in the AlphaGo series is generally believed to be due to the assistance of MCTS and deep learning. We will then verify the learning effect of alpha-T, whether it can learn useful paths and find the right nodes efficiently. First, we verified the accuracy rate of the agent to find the target node with the increasing number of training episodes on TSP20/50/100, and then we compared with baselines the change of learning convergence with the increasing number of training steps.

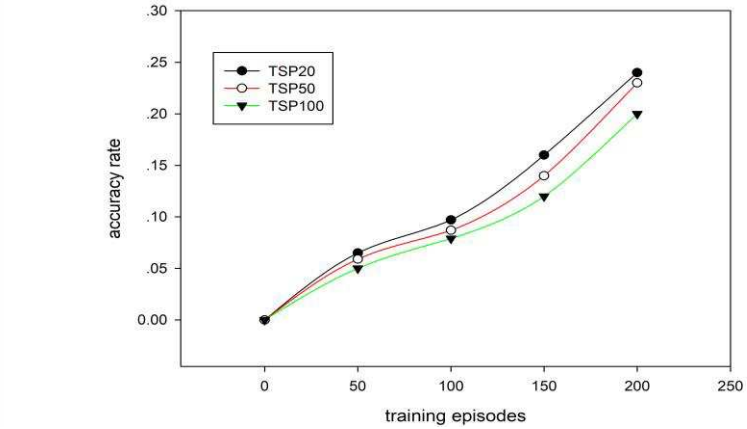

Fig. 5. It shows the accuracy of finding the right node with the increasing number of training episodes.

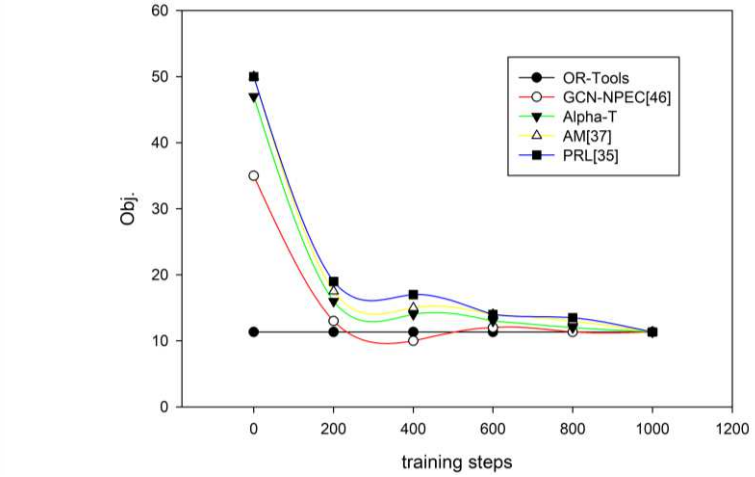

Fig. 6. Comparison of Alpha-T's learning curve with baselines on VRP100 as the number of training steps increase.

From Figure 5, we can see that the accuracy of finding the right node increases with the increasing number of training steps of agents, which indicates that the learning and updating process of Q-learning with MCTS is effective. It can be seen from Figure 6 that Alpha-T gradually converges to the optimal solution with the increase of training steps, and its effect is obvious at the beginning stage, which indicates the effectiveness and efficiency of gradient descent. Because GCN-NPEC employs supervised learning in the initial stage, it converges faster in the initial stage.

5 Discussion and Conclusion

We designed alpha-T inspired by AlphaZero, which does not require any expert experience and can learn heuristics entirely from unlabeled data through self-play. Aiming at path optimization problems (including TSP, VRP, etc.), we have studied a lightweight self-playing framework based on MCTS, whose consumption is far lower than AlphaZero. Since the memory component can be targeted to record the history trajectories to assist the modeling of the policy network and value network later, this is equivalent to saving a lot of computing resources in the early stage and avoiding the supervised pre-training in AlphaGo. We apply the Q-learning algorithm with MCTS to train the policy network and the value network together so that both can be iteratively updated at the same time to produce the best model, which simultaneously effectively integrates learners, data generators, and model evaluators. Like AlphaZero, which plays not just Go but a variety of board games, Alpha-T can be generalized to other similar combinatorial optimization problems on a much larger scale. It also shows that combinatorial optimization is one of the most cutting-edge problems in the field of artificial intelligence, and its scientific value and practical application value make us try to apply state-of-the-art technology to it. In this paper, we propose a gen-

eral RL framework in which components can be replaced by other models, such as the state-of-the-art GNNs and RL, which will be our work in the future.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No.61671157.

Ethical Standards statements

Our study did not raise any ethical questions, i.e. none of the subjects were humans or living individuals. This paper only focuses on combinatorial optimization of graphs in computer science, and the technologies used are all modern computer technologies, including deep learning, reinforcement learning, and Monte Carlo tree search. Our research belongs to theoretical innovation and application innovation in computer science, so it does not involve any ethical and moral issues.

Informed Consent

All authors of this article are aware of this article and agree to its submission.

Conflict of interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He/she is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from 17110240039@fudan.edu.cn.

Signed by all authors as follows:

References

1. Ye, J., Zhao, J., Ye, K., Xu, C.: How to build a graph-based deep learning architecture in traffic domain: A survey. arXiv. 1–21 (2020).
2. Mor, A., Speranza, M.G.: Vehicle routing problems over time: a survey. 4or. 18, 129–149 (2020). https://doi.org/10.1007/s10288-020-00433-2.
3. Goyal, S.: A Survey on Travelling Salesman Problem. Midwest Instr. Comput. Symp. 1–9 (2010).
4. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. Adv. Neural Inf. Process. Syst. 2017-Decem, 6349–6359 (2017).
5. Hamrick, J.B., Mohamed, S.: Levels of Analysis for Machine Learning. arXiv. 1–6 (2020).
6. Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature. 521, 436–444 (2015). https://doi.org/10.1038/nature14539.
7. Mousavi, S.S., Schukat, M., Howley, E.: Deep Reinforcement Learning: An Overview. Lect. Notes Networks Syst. 16, 426–440 (2018). https://doi.org/10.1007/978-3-319-56991-8_32.
8. Botvinick, M., Ritter, S., Wang, J.X., Kurth-Nelson, Z., Blundell, C., Hassabis, D.: Reinforcement Learning, Fast and Slow. Trends Cogn. Sci. 23, 408–422 (2019). https://doi.org/10.1016/j.tics.2019.02.006.
9. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'Horizon. arXiv. 1–47 (2018).
10. Yang, Y., Whinston, A.: A Survey on Reinforcement Learning for Combinatorial Optimization. arXiv. 4165, 0–2 (2020).
11. Vesselinova, N., Steinert, R., Perez-Ramirez, D.F., Boman, M.: Learning Combinatorial Optimization on Graphs: A Survey with Applications to Networking. IEEE Access. 8, 120388–120416 (2020). https://doi.org/10.1109/ACCESS.2020.3004964.
12. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature. 529, 484–489 (2016). https://doi.org/10.1038/nature16961.

13. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of Go without human knowledge. Nature. 550, 354–359 (2017). https://doi.org/10.1038/nature24270.
14. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science (80-.). 362, 1140–1144 (2018). https://doi.org/10.1126/science.aar6404.
15. Eds, A.L.: Mauro Birattari Tuning Metaheuristics : A Machine Learning Perspective Studies in Computational Intelligence , Volume 197.
16. Ivanov, S., D'yakonov, A.: Modern Deep Reinforcement Learning Algorithms. arXiv. (2019).
17. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games. 4, 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810.
18. Velicković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. arXiv. 1–12 (2017).
19. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf. 1724–1734 (2014). https://doi.org/10.3115/v1/d14-1179.
20. Hopfield, J.J., Tank, D.W.: "Neural" computation of decisions in optimization problems. Biol. Cybern. 52, 141–152 (1985). https://doi.org/10.1007/BF00339943.
21. Jordan, M.I., Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects. 349, (2015).
22. Yang, Y., Whinston, A.: A Survey on Reinforcement Learning for Combinatorial Optimization. arXiv. (2020).
23. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'Horizon. arXiv. 1–34 (2018).
24. Littman, M.L.: Reinforcement learning improves behaviour from evaluative feedback. Nature. 521, 445–451 (2015). https://doi.org/10.1038/nature14540.
25. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature. 518, 529–533 (2015). https://doi.org/10.1038/nature14236.
26. Hafner, D., Lillicrap, T., Norouzi, M., Ba, J.: Mastering Atari with Discrete World Models. (2020).
27. Badia, A.P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., Blundell, C.: Agent57: Outperforming the atari human benchmark. arXiv. (2020).
28. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.:

Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv. 1–19 (2017).
29. Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., Silver, D.: Learning to search with MCTSnets. 35th Int. Conf. Mach. Learn. ICML 2018. 4, 2920–2931 (2018).
30. Jaderberg, M., Czarnecki, W.M., Dunning, I., Marris, L., Lever, G., Castaneda, A.G., Beattie, C., Rabinowitz, N.C., Morcos, A.S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J.Z., Silver, D., Hassabis, D., Kavukcuoglu, K., Graepel, T.: Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. arXiv. 865, 859–865 (2018).
31. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J.P., Jaderberg, M., Vezhnevets, A.S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T.L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., Silver, D.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature. 575, 350–354 (2019). https://doi.org/10.1038/s41586-019-1724-z.
32. Sutskever, I., Vinyals, O., Le, Q. V.: Sequence to sequence learning with neural networks. Adv. Neural Inf. Process. Syst. 4, 3104–3112 (2014).
33. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Adv. Neural Inf. Process. Syst. 2015-Janua, 2692–2700 (2015).
34. Bello, I., Pham, H., Le, Q. V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. 5th Int. Conf. Learn. Represent. ICLR 2017 - Work. Track Proc. 1–15 (2017).
35. Nazari, M., Oroojlooy, A., Takáč, M., Snyder, L. V.: Reinforcement learning for solving the vehicle routing problem. Adv. Neural Inf. Process. Syst. 2018-Decem, 9839–9849 (2018).
36. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5999–6009 (2017).
37. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! 7th Int. Conf. Learn. Represent. ICLR 2019. 1–25 (2019).
38. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. arXiv. 1–38 (2018).
39. Xu, K., Jegelka, S., Hu, W., Leskovec, J.: How powerful are graph neural networks? 7th Int. Conf. Learn. Represent. ICLR 2019. (2019).
40. Cui, P., Wang, X., Pei, J., Zhu, W.: A Survey on Network Embedding. IEEE Trans. Knowl. Data Eng. 31, 833–852 (2019). https://doi.org/10.1109/TKDE.2018.2849727.

41. Wu, F., Zhang, T., de Souza, A.H., Fifty, C., Yu, T., Weinberger, K.Q.: Simplifying graph convolutional networks. 36th Int. Conf. Mach. Learn. ICML 2019. 2019-June, 11884–11894 (2019).
42. Li, Z., Chen, Q., Koltun, V.: Combinatorial optimization with graph convolutional networks and guided tree search. Adv. Neural Inf. Process. Syst. 2018-Decem, 539–548 (2018).
43. Mittal, A., Dhawan, A., Manchanda, S., Medya, S., Ranu, S., Singh, A.: Learning heuristics over large graphs via deep reinforcement learning. arXiv. (2019).
44. Joshi, C.K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. arXiv. 1–17 (2019).
45. Drori, I., Kates, B., Kharkar, A., Ma, Q., Sickinger, W.R., Ge, S., Dolev, E., Dietrich, B., Williamson, D.P., Udell, M.: Learning to Solve Combinatorial Optimization Problems on Real-World Graphs in Linear Time. arXiv. 1–19 (2020).
46. Duan, L., Zhan, Y., Hu, H., Gong, Y., Wei, J., Zhang, X., Xu, Y.: Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach. Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. 3054–3063 (2020). https://doi.org/10.1145/3394486.3403356.
47. Ma, Q., Ge, S., He, D., Thaker, D., Drori, I.: Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. arXiv. (2019).
48. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. Adv. Neural Inf. Process. Syst. 32, (2019).
49. Sobieczky, H.: a Learning-Based Iterative Method for Solving Vehicle Routing Problems. Iclr. 3, 3–5 (2020).
50. Huang, J., Patwary, M., Diamos, G.: Coloring big graphs with AlphaGoZero. arXiv. (2019).
51. Xing, Z., Tu, S., Xu, L.: Solve traveling salesman problem by Monte Carlo tree search and deep neural network. arXiv. (2020).
52. Abe, K., Xu, Z., Sato, I., Sugiyama, M.: Solving NP-Hard Problems on Graphs with Extended AlphaGo Zero. 1–23 (2019).
53. Laterre, A., Cohen, A.S., Chen, H., Fu, Y., Kas, D., Dahl, T.S., Beguir, K., Jabri, M.K., Hajjar, K., Kerkeni, A.: Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. arXiv. (2018).
54. Zhang, J., Cui, L., Gouza, F.B.: SeGEN: Sample-ensemble genetic evolutionary network model. arXiv. 1–12 (2018).
55. Ma, X., Zhu, Q., Zhou, Y., Li, X., Wu, D.: Improving question generation with sentence-level semantic matching and answer position inferring. arXiv. (2019). https://doi.org/10.1609/aaai.v34i05.6366.
56. Wang, G., Ying, R., Huang, J., Leskovec, J.: Direct Multi-hop Attention based Graph Neural Network. 1–15 (2020).
57. Hinton, G., Vinyals, O., Dean, J.: Distilling the Knowledge in a Neural Network. 1–9 (2015).
58. Jin, C., Allen-Zhu, Z., Bubeck, S., Jordan, M.I.: Is Q-learning provably efficient? Adv. Neural Inf. Process. Syst. 2018-Decem, 4863–4873 (2018).
59. Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. Adv. Neural Inf. Process. Syst. 2017-Decem, 5361–5371 (2017).

60. Tian, Y., Ma, J., Gong, Q., Sengupta, S., Chen, Z., Pinkerton, J., Lawrence Zitnick, C.: Elf OpenGo: An analysis and open reimplementation of Alphazero. In: 36th International Conference on Machine Learning, ICML 2019. pp. 10885–10894 (2019).
61. Wu, T.R., Wei, T.H., Wu, I.C.: Accelerating and Improving AlphaZero Using Population Based Training, (2020). https://doi.org/10.1609/aaai.v34i01.5454.
62. Zhang, J., Zhang, H., Xia, C., Sun, L.: Graph-Bert: Only Attention is Needed for Learning Graph Representations. (2020).

Figures

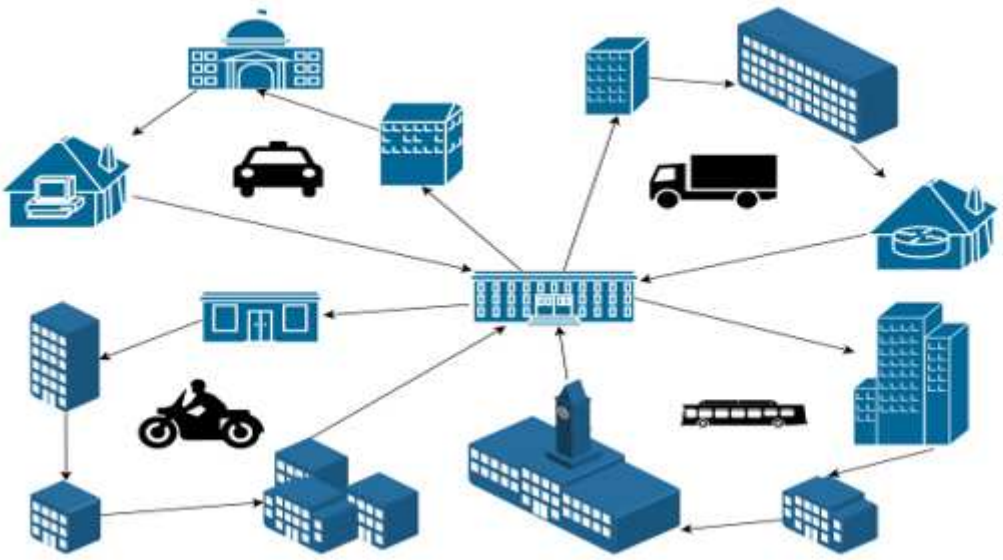

Figure 1

The demonstration of VRP in the practical application scenario, in which each node can be different facilities or cities containing their demand; Different paths correspond to different miles; A vehicle can be a truck, a motorbike, a taxi or a shuttle bus with their respective load capacity. The goal of VRP is to optimize a set of paths to minimize the total cost.

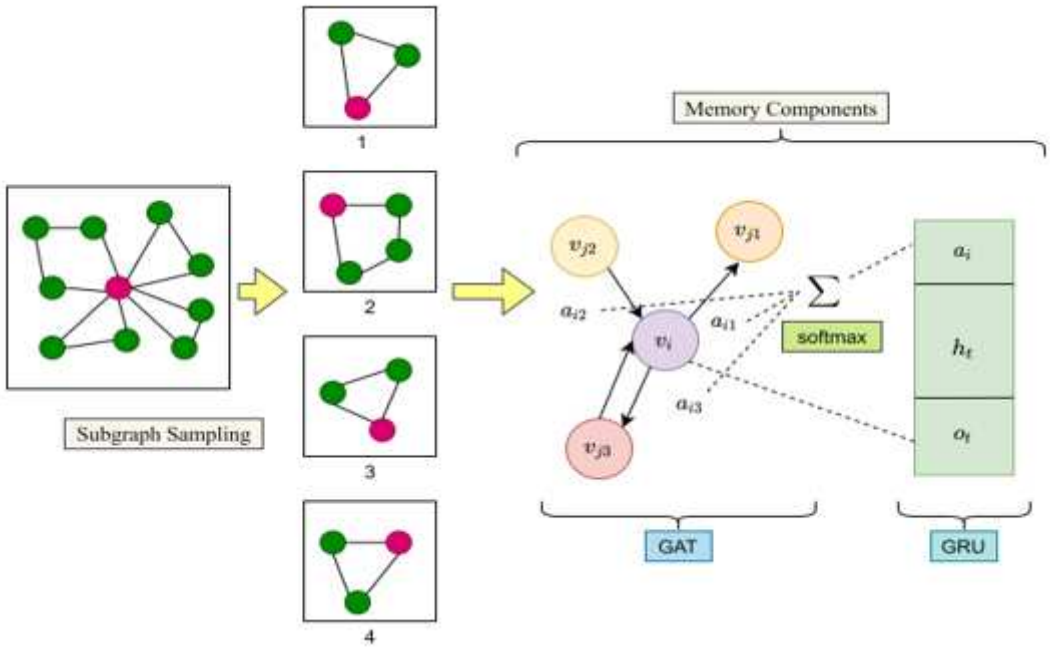

Figure 2

Flowchart of memory components with GAT and GRU. We take the graph as input and apply subgraph sampling to divide it into k subgraphs with central nodes, then iteratively process each subgraph using the memory component. The subgraph sampling process in the figure is just a simple example, while the actual graph is much more complex. The memory component combining GAT and GRU can effectively infer the importance of neighbor nodes and remember and update their status and trajectories, which enables the model to avoid pre-training.

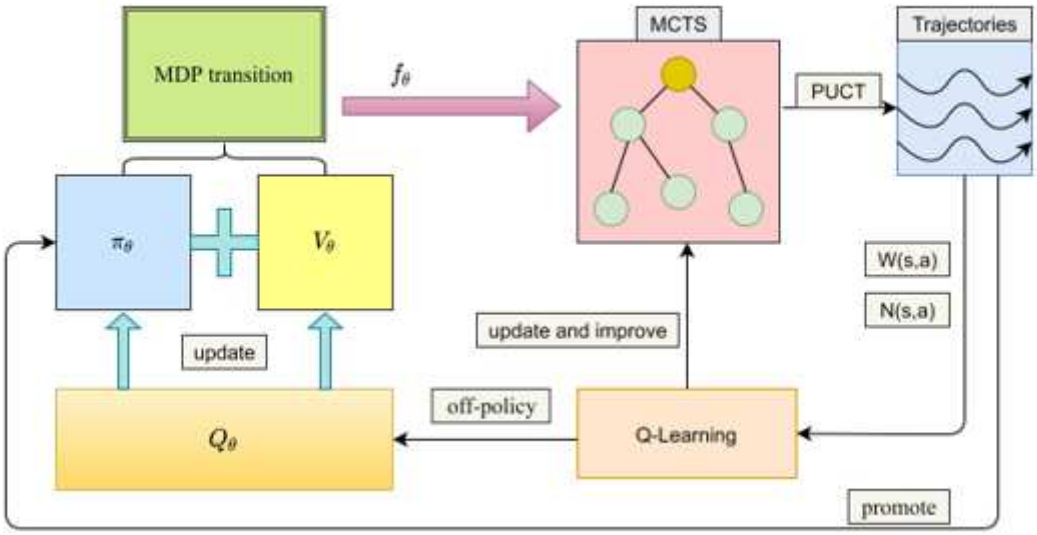

Figure 3

Q-learning with MCTS is applied to simultaneously model and train the policy network and the value network.

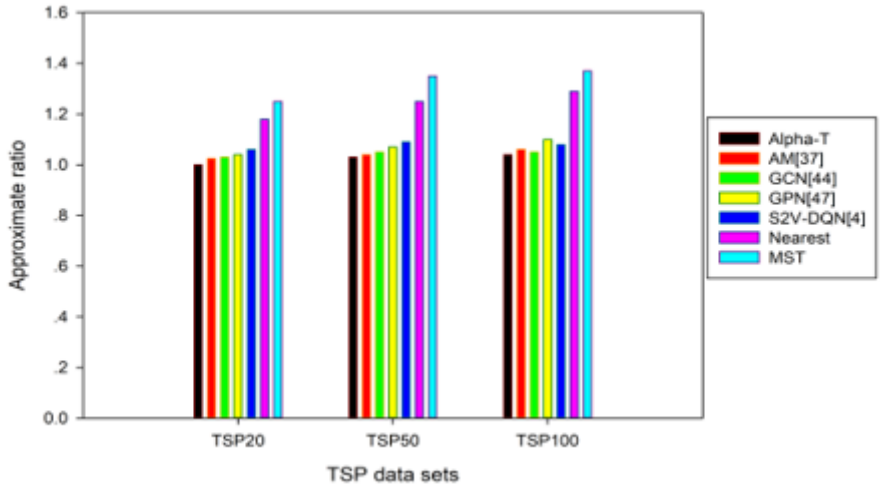

Figure 4

The approximate ratio of the solutions generated by different methods on small TSP in-stances to the optimal solutions.

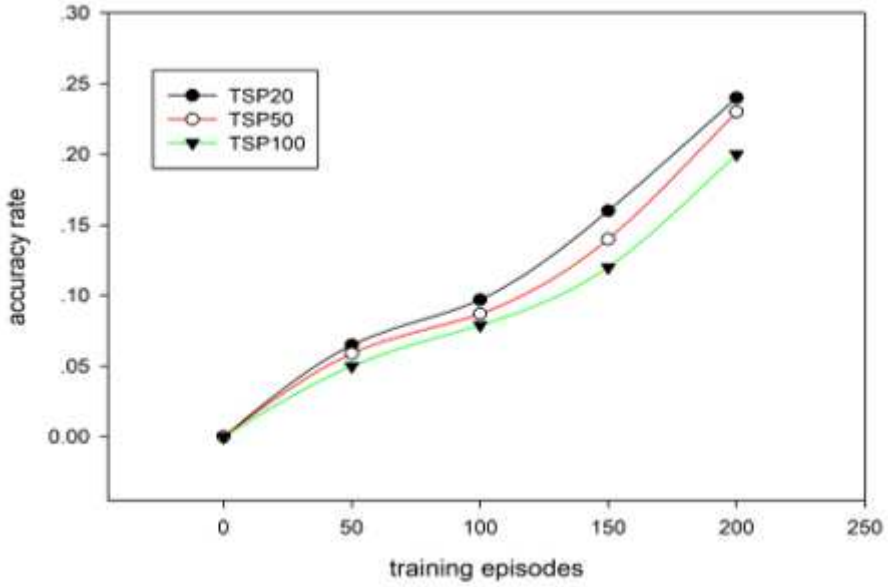

Figure 5

It shows the accuracy of finding the right node with the increasing number of training epi-sodes.

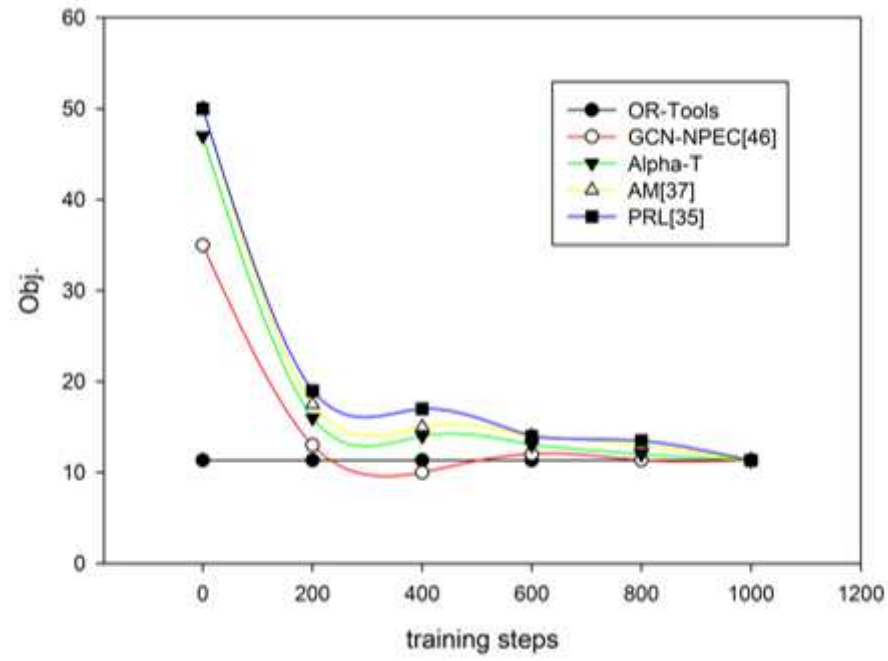

Figure 6

Comparison of Alpha-T's learning curve with baselines on VRP100 as the number of train-ing steps increase.